A

# From almost optimal algorithms to logics for complexity classes via listings and a halting problem

YIJIA CHEN, Shanghai Jiaotong University
JÖRG FLUM, Albert-Ludwigs-Universität Freiburg

Let C denote one of the complexity classes "polynomial time," "logspace," or "nondeterministic logspace." We introduce a logic $L(\mathrm{C})_{\mathrm{inv}}$ and show generalizations and variants of the equivalence $\big(L(\mathrm{C})_{\mathrm{inv}}$ captures C if and only if there is an almost C-optimal algorithm in C for the set TAUT of tautologies of propositional logic.$\big)$ These statements are also equivalent to the existence of a listing of subsets in C of TAUT by corresponding Turing machines and equivalent to the fact that a certain parameterized halting problem is in the parameterized complexity class $\mathrm{XC}_{\mathrm{uni}}$.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*model theory*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*complexity of proof procedures*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Optimal algorithms, logics for complexity classes, listings, parameterized complexity

## 1. INTRODUCTION

As the title already indicates, this paper relates two topics which at first glance seem to be unrelated. On the one hand we consider almost optimal algorithms. An algorithm, say deciding the class TAUT of tautologies of propositional logic, is *almost optimal* if the time it requires to accept tautologies can be polynomially bounded in terms of the corresponding time of any other algorithm deciding TAUT.[1] In their fundamental paper [Krajíček and Pudlák 1989] Krajíček and Pudlák not only introduced the notion of almost optimality but they also derived a series of statements equivalent to the existence of an almost optimal algorithm for TAUT, among them the existence of a polynomially optimal propositional proof system. Furthermore, they stated the conjecture:

*Conjecture 1.* There is no almost optimal algorithm for TAUT.

On the other hand, the question of whether there is a logic capturing the complexity class P (polynomial time) remains the central open problem in descriptive complexity. By a result due to Immerman and Vardi [Immerman 1986; Vardi 1982] least fixed-point logic LFP captures P on *ordered* structures. There are artificial logics capturing P (on arbitrary structures), but they do not fulfill a natural requirement to logics in this context:

> There is an algorithm which decides whether $\mathcal{A}$ is a model of $\varphi$ for all structures $\mathcal{A}$ and sentences $\varphi$ of the logic and which, for fixed $\varphi$, has running time polynomial in the size of $\mathcal{A}$. (1)

If this condition is fulfilled for a logic capturing polynomial time, we speak of a P-bounded logic for P. In [Gurevich 1988] Gurevich stated the conjecture:

*Conjecture 2.* There is no P-bounded logic for P.

The conjecture is false if one waives the effectivity condition (1). This is shown in [Gurevich 1988, Section 7, CLAIM 2] by considering a logic, *the order-invariant least*

[1] All notions will be defined in a precise manner later.

*fixed-point logic*, introduced by Blass and Gurevich and which we denote by $\mathrm{LFP}_{\mathrm{inv}}$.[2] For any vocabulary the sentences of $\mathrm{LFP}_{\mathrm{inv}}$ are the sentences of least fixed-point logic LFP in a vocabulary with an additional binary relation symbol for orderings. In $\mathrm{LFP}_{\mathrm{inv}}$ for a structure $\mathcal{A}$ to be a model of $\varphi$ it is required that in all structures of cardinality less than or equal to that of $\mathcal{A}$, the validity of $\varphi$ (as a sentence of least fixed-point logic) does not depend on the chosen ordering, and $\mathcal{A}$ with some ordering satisfies $\varphi$. As $\mathrm{LFP}_{\mathrm{inv}}$ satisfies all requirements of a P-bounded logic for P except (1), Gurevich implicitly states the conjecture:

*Conjecture 2*.* $\mathrm{LFP}_{\mathrm{inv}}$ is not a P-bounded logic for P.

We show that

$$\text{Conjecture 1 is true} \iff \text{Conjecture 2* is true.} \tag{2}$$

In general, the experts tend to believe Conjecture 1, as the existence of an almost optimal algorithm for TAUT would have various consequences which seem to be unlikely (see [Köbler et al. 2003; Krajíček and Pudlák 1989]). It is worthwhile to emphasize that we show that Conjecture 1 is equivalent to Conjecture 2* and do not claim its equivalence to Conjecture 2. The situation with Conjecture 2 is quite different; no known consequences of the existence of a P-bounded logic for P seem to be implausible. Moreover, due to results showing that there are logics capturing polynomial time on always larger classes of structures, Grohe [Grohe 2009] "mildly leans towards believing" that there is a P-bounded logic for P.

We mentioned that at first glance "almost optimal algorithms for TAUT" and "logics for P" seem to be unrelated topics. However, there are reformulations of Conjecture 1 and Conjecture 2 that are alike. In fact, it is known [Sadowski 2002] that TAUT has an almost optimal algorithm if and only if there is an effective enumeration or *listing* of all subsets of TAUT that are in P by means of polynomial time Turing machines that decide them. And it is not hard to see that there is a P-bounded logic for P if and only if there is a listing of all polynomial time decidable classes of graphs closed under isomorphism, again a listing in terms of polynomial time Turing machines that decide these classes. In fact the question for a logic for P was stated in this way by Chandra and Harel [Chandra and Harel 1982] in the context of an analysis of the complexity and expressiveness of query languages. Hence the equivalence (2) can be reformulated as:

$$\begin{aligned}&\mathrm{LFP}_{\mathrm{inv}} \text{ is a P-bounded logic for P} \iff \\ &\qquad\qquad\qquad\qquad \text{there is a listing of the subsets in P of TAUT.}\end{aligned} \tag{3}$$

And one consequence of (2) is:

> *If there is a listing of the subsets in* P *of* TAUT*, then there is a listing of the polynomial time decidable classes of graphs closed under isomorphism.*

The reformulation (3) led us to the idea underlying the proof of the implication from left to right: We express the property of a propositional formula $\alpha$ of being a tautology in $\mathrm{LFP}_{\mathrm{inv}}$ by reducing the second-order quantifier in "all assignments satisfy $\alpha$" to the second-order quantifier "for all orderings" hidden in the definition of the satisfaction relation of $\mathrm{LFP}_{\mathrm{inv}}$; then a listing of the appropriate sentences of $\mathrm{LFP}_{\mathrm{inv}}$ yields a listing of the subsets in P of TAUT.

Later on we realized that one gets a listing of the subsets in P of TAUT if one assumes that there is a listing of its subsets in L (logarithmic space). There are standard

[2] In [Gurevich 1988] the logic is defined in a slightly different form.

logics DTC (deterministic transitive closure logic) and TC (transitive closure logic) capturing, on *ordered* structures, the complexity classes L and NL (nondeterministic logarithmic space), respectively [Immerman 1987; 1988]. We realized that for their corresponding order-invariant logics $\mathrm{DTC}_{\mathrm{inv}}$ and $\mathrm{TC}_{\mathrm{inv}}$ the analogues of the equivalence (3) hold. Therefore, by the result on listings just mentioned, $\mathrm{LFP}_{\mathrm{inv}}$ captures P if $\mathrm{DTC}_{\mathrm{inv}}$ captures L. Note that it is not known whether the existence of a logic capturing P is implied by the existence of a logic capturing L.

A more general notion of listing turned out to be helpful. For complexity classes C and C$'$ we consider listings of the C-subsets of TAUT (that are, subsets in C of TAUT) by means of Turing machines of type C$'$; we write $\textsc{List}(\mathrm{C}, \textsc{Taut}, \mathrm{C}')$ if such a listing exists. Here, C and C$'$ range over the complexity classes L, NL, P, and NP. For the classes P and NP such listings were already considered and put to good use by Sadowski in [Sadowski 2002]. This more general notion referring to two complexity classes is also meaningful in the context of logics. If we say that a logic is a P-bounded logic for P, the second "P" refers to the classes axiomatizable in the logic and the first one to the polynomial time property expressed in (1). This suggests the definition of a C$'$*-bounded logic for* C. It turns out that these general concepts of listings and logics match. In fact we get for $\mathrm{C} \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$, $\mathrm{C}' \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}\}$ with $\mathrm{C} \subseteq \mathrm{C}'$

$$L(\mathrm{C}) \textit{ is a } \mathrm{C}'\textit{-bounded logic for } \mathrm{C} \iff \textsc{List}(\mathrm{C}, \textsc{Taut}, \mathrm{C}'). \tag{4}$$

Here $L(\mathrm{C})$ is $\mathrm{DTC}_{\mathrm{inv}}$, $\mathrm{TC}_{\mathrm{inv}}$, and $\mathrm{LFP}_{\mathrm{inv}}$ if C is L, NL, and P, respectively. This relationship between listings and logics is not only fruitful for the side of the logics but also for the side of listings. For example, we get

$$\textit{If } \textsc{List}(\mathrm{L}, \textsc{Taut}, \mathrm{L}), \textit{ then } \textsc{List}(\mathrm{NL}, \textsc{Taut}, \mathrm{NL}). \tag{5}$$

As shown in [Sadowski 2002], the property $\textsc{List}(\mathrm{P}, \textsc{Taut}, \mathrm{P})$ is equivalent to the existence of an almost optimal algorithm for TAUT $\big($by (2) and (3)$\big)$ and, as already mentioned, to the existence of a polynomially optimal propositional proof system. We show the analogues of these equivalences for L and space optimality instead of P and time optimality. Moreover, we do this not only for TAUT but for arbitrary problems $Q$ with padding. In particular, by (5), we get the following, perhaps surprising relationship between space optimal and time optimal algorithms:

> *Assume $Q$ has padding. If $Q$ has an almost space optimal algorithm, then it has an almost (time) optimal algorithm.*

While listings were the main tool for proving the implication from left to right of (2), a halting problem plays the role of a bridge leading to the converse direction. In [Nash et al. 2005] Nash, Remmel, and Vianu have raised the question whether one can *prove* Conjecture 2*. They give a reformulation of this conjecture in terms of the complexity of a halting problem for nondeterministic Turing machines. This reformulation is best expressed in the terminology of parameterized complexity. We introduce the following parameterized halting problem $p$-$\textsc{Halt}_{>}$ for nondeterministic Turing machines.

$p$-$\textsc{Halt}_{>}$

| | |
|---|---|
| *Instance:* | A nondeterministic Turing machine $\mathbb{M}$ and $\overbrace{1\ldots1}^{n}$ with $n \in \mathbb{N}$. |
| *Parameter:* | $\lvert\mathbb{M}\rvert$, the size of $\mathbb{M}$. |
| *Question:* | Does every accepting run of $\mathbb{M}$ on the empty input tape take more than $n$ steps? |

Then, by [Nash et al. 2005],

$$\mathrm{LFP}_{\mathrm{inv}} \text{ is a P-bounded logic for P} \iff p\text{-}\textsc{Halt}_{>} \in \mathrm{XP}_{\mathrm{uni}}. \tag{6}$$

Here $\mathrm{XP}_{\mathrm{uni}}$ denotes a parameterized complexity class (to be defined in Section 2). As TAUT the classical problem underlying $p$-$\textsc{Halt}_{>}$ is coNP-complete. Based on this fact we can show that the existence of an almost optimal algorithm for TAUT implies (even is equivalent to) $p$-$\textsc{Halt}_{>} \in \mathrm{XP}_{\mathrm{uni}}$, and thereby we get a proof of the missing implication of (2).

It seems to be hard to get reasonable upper and lower bounds for the complexity of $p$-$\textsc{Halt}_{>}$ by showing its (non-)membership in one of the standard classes of parameterized complexity like $\mathrm{FPT}_{\mathrm{uni}}$, $\mathrm{XP}_{\mathrm{uni}}$, $\mathrm{XL}_{\mathrm{uni}}$, $\mathrm{XNP}_{\mathrm{uni}}$, FPT, XP, XNP, .... However, for every of these classes there is a natural extension of (6) (for the classes $\mathrm{FPT}_{\mathrm{uni}}, \mathrm{XP}_{\mathrm{uni}}, \mathrm{FPT}$, and XP they were already derived in [Chen and Flum 2010a] and [Nash et al. 2005]). These equivalences lead to extensions of the equivalence (2) and more importantly, to interesting notions of strongly and of effectively almost optimal algorithms. An almost optimal algorithm for $Q$ is effectively almost optimal if from any other algorithm $\mathbb{A}$ deciding $Q$ we can *compute* a polynomial bounding its running time in terms of the running time of $\mathbb{A}$. It is strongly almost optimal if the bounding polynomial can always be chosen of the same degree. In [Krajíček and Pudlák 1989] the authors show that an almost optimal algorithm for TAUT exists, if NE = E. We can even derive

> *If* NE = E*, then* TAUT *has an effectively and strongly almost optimal algorithm.*

On the other hand we have

> *If* NP[TC] $\neq$ P[TC]*, then* TAUT *has no effectively and strongly almost optimal algorithm.*

Here NP[TC] $\neq$ P[TC] is an extension of the hypothesis NP $\neq$ P introduced in [Chen and Flum 2010d].

Let us close these introductory remarks by presenting explicitly one of the results hidden in the previous exposition that relates the four concepts mentioned in the title of this paper.

> *The following are equivalent:*
> – TAUT *has an almost space optimal algorithm.*
> – *The logic* $\mathrm{DTC}_{\mathrm{inv}}$ *is* L*-bounded for* L.
> – $\textsc{List}(\mathrm{L}, \textsc{Taut}, \mathrm{L})$.
> – $p$-$\textsc{Halt}_{>} \in \mathrm{XL}_{\mathrm{uni}}$.

The content of the different sections is the following. After recalling some basic notions in Section 2, we turn to a proof of statements relating the existence of almost optimal algorithms for TAUT with the complexity of $p$-$\textsc{Halt}_{>}$ in Section 3. In Section 4 we relate this complexity to capturing properties of the different invariant logics. Part of the corresponding proof is postponed to Section 5, which is devoted to so-called slicewise downward monotone parameterized problems. This concept helps us to show that it is as complex to check the order-invariance of first-order sentences as it is to check that of LFP-sentences. In Section 6 we extend the known relationship between almost optimal algorithms, polynomially optimal propositional proof systems, and listings to different variants (strong, effective, logarithmic space) of these concepts. Finally, in Section 7 we prove the relationship (4) between logics and listings.

This paper is an extended version of the papers [Chen and Flum 2010b; 2010c; 2011].

## 2. SOME PRELIMINARIES

In this section we fix some notations and recall some basic definitions and concepts from parameterized complexity.

We let $\mathbb{N}[X]$ and $\mathbb{N}_d[X]$, where $d \in \mathbb{N}$, be the set of polynomials and the set of polynomials of degree $\leq d$, respectively, with natural numbers as coefficients. We denote the alphabet $\{0,1\}$ by $\Sigma$ and the length of a string $x \in \Sigma^*$ by $|x|$. Let $1^n$ be the string consisting of $n$ 1s and let $\lambda$ denote the empty string. We identify problems with subsets $Q$ of $\Sigma^*$. Sometimes statements containing a formulation like "there is a $d \in \mathbb{N}$ such that for all $x \in \Sigma^*$: $\ldots \leq |x|^d$" or containing a term $\log |x|$ can be wrong for $x \in \Sigma^*$ with $|x| \leq 1$. We trust the reader's common sense to interpret such statements reasonably.

We denote by P and L the classes of problems $Q$ such that $x \in Q$ is solvable by a deterministic Turing machine in time polynomial in $|x|$ and in space $O(\log |x|)$, respectively. The corresponding nondeterministic classes are NP and NL. *In this paper,* C*,* C$'$*,* C$_0$*, . . . will always denote one of the complexity classes* L*,* P*,* NL*, and* NP.

A problem $Q \subseteq \Sigma^*$ *has padding* if there is a function $\mathit{pad} : \Sigma^* \times \Sigma^* \to \Sigma^*$ computable in logarithmic space having the following properties:

(i) For any $x, y \in \Sigma^*$, $|\mathit{pad}(x,y)| > |x| + |y|$ and $\big(\mathit{pad}(x,y) \in Q \iff x \in Q\big)$.
(ii) There is a logspace algorithm which, given $\mathit{pad}(x,y)$ recovers $y$.

By $\langle \ldots \rangle$ we denote some standard logspace and linear time computable tupling function with logspace and linear time computable inverses. *In this paper we always assume that $Q$ denotes a decidable and nonempty problem.*

If $\mathbb{A}$ is a (deterministic or nondeterministic) algorithm and $\mathbb{A}$ accepts $x \in \Sigma^*$, then we denote by $t_{\mathbb{A}}(x)$ the number of steps of a shortest accepting run of $\mathbb{A}$ on $x$; if $\mathbb{A}$ does not accept $x$, then $t_{\mathbb{A}}(x) := \infty$. By convention, $n < \infty$ for $n \in \mathbb{N}$. Similarly, $s_{\mathbb{A}}(x)$ is the minimum of the space used by accepting runs on $x$. *By default, algorithms are deterministic*. If an algorithm $\mathbb{A}$ on input $x$ eventually halts and outputs a value, we denote it by $\mathbb{A}(x)$.

We use deterministic and nondeterministic Turing machines with $\Sigma$ as alphabet as our basic computational model for algorithms (and we often use the notions "algorithm" and "Turing machine" synonymously). If necessary we will not distinguish between a Turing machine and its code, a string in $\Sigma^*$. If $\mathbb{M}$ is a deterministic or nondeterministic Turing machine, then $L(\mathbb{M})$ is the language accepted by $\mathbb{M}$. We use Turing machines as acceptors and transducers. Even though we use formulations like "let $\mathbb{M}_1, \mathbb{M}_2, \ldots$ be an enumeration of *all* polynomial time Turing machines," from the context it will be clear that we only refer to acceptors (or that we only refer to transducers). We assume that a run of a nondeterministic Turing machine is determined by the sequence of its states.

### 2.1. Parameterized complexity

We view *parameterized problems* as pairs $(Q, \kappa)$ consisting of a classical problem $Q \subseteq \Sigma^*$ and a *parameterization* $\kappa : \Sigma^* \to \mathbb{N}$, which is required to be polynomial time computable. We will present parameterized problems in the form we did it for $p$-HALT$_>$ in the Introduction.

A parameterized problem $(Q, \kappa)$ is in the class FPT$_{\text{uni}}$ (or *uniformly fixed-parameter tractable*) if $x \in Q$ is solvable by a deterministic algorithm running in time $\leq f(\kappa(x)) \cdot |x|^{O(1)}$ for some $f : \mathbb{N} \to \mathbb{N}$.

Let C $\in \{\text{L}, \text{NL}, \text{P}, \text{NP}\}$. A parameterized problem $(Q, \kappa)$ is in the class XC$_{\text{uni}}$ if there is a deterministic (nondeterministic) algorithm deciding (accepting) $Q$ and witnessing

for every $k \in \mathbb{N}$ that the classical problem

$$(Q, \kappa)_k := \big\{ x \in Q \;\big|\; \kappa(x) = k \big\},$$

the $k$th *slice* of $(Q, \kappa)$, is in C. For example, $(Q, \kappa)$ is in the class $\mathrm{XP}_{\mathrm{uni}}$ if there is a deterministic algorithm $\mathbb{A}$ deciding $x \in Q$ in time $|x|^{f(\kappa(x))}$ for some function $f : \mathbb{N} \to \mathbb{N}$. And $(Q, \kappa)$ is in the class $\mathrm{XNP}_{\mathrm{uni}}$ if there is a nondeterministic algorithm $\mathbb{A}$ accepting $Q$ such that for some function $f : \mathbb{N} \to \mathbb{N}$ we have $t_{\mathbb{A}}(x) \leq |x|^{f(\kappa(x))}$ for all $x \in Q$.

We have added the subscript "uni" to the names of these classes to emphasize that they are classes of the so-called uniform parameterized complexity theory. If in the definition of $\mathrm{FPT}_{\mathrm{uni}}$, $\mathrm{XP}_{\mathrm{uni}}$, and $\mathrm{XNP}_{\mathrm{uni}}$ we require the function $f$ to be computable, then we get the corresponding classes FPT, XP, and XNP of the strongly uniform theory. Now the interested reader will have no difficulties to define the classes XL and XNL, classes we do not use in this paper. For further information on parameterized complexity we refer to [Downey and Fellows 1999; Flum and Grohe 2006].

## 3. ALMOST OPTIMAL ALGORITHMS AND $P$-HALT$_>$

In this section we relate the existence of an almost optimal algorithm for the set TAUT of tautologies of propositional logic to the complexity of the parameterized problem $p$-HALT$_>$.

Recall that $\mathbb{N}_d[X]$ is the set of polynomials of degree $\leq d$ and that $t_{\mathbb{A}}(x)$ denotes the minimum of the running times of accepting runs of the algorithm $\mathbb{A}$ on input $x$.

DEFINITION 3.1. *An algorithm $\mathbb{O}$ deciding a problem $Q \subseteq \Sigma^*$ is* almost optimal *if for every algorithm $\mathbb{A}$ deciding $Q$ there is a polynomial $p \in \mathbb{N}[X]$ such that*

$$t_{\mathbb{O}}(x) \leq p(t_{\mathbb{A}}(x) + |x|) \tag{7}$$

*for all $x \in Q$. Note that nothing is required for $x \notin Q$. If there is a $d \in \mathbb{N}$ such that the polynomial $p$ can be chosen in $\mathbb{N}_d[X]$ for all $\mathbb{A}$, then $\mathbb{O}$ is* strongly almost optimal.

We shall need the following fact.

LEMMA 3.2. *Let $Q$ be polynomial time reducible to $Q'$ and assume that $Q'$ has padding. If $Q'$ has a (strongly) almost optimal algorithm, then so does $Q$.*

*Proof:* As $Q'$ has padding, there is a one-to-one polynomial time reduction $\mathbb{S}$ from $Q$ to $Q'$ with a polynomial time inverse. Let $\mathbb{O}'$ be a (strongly) almost optimal algorithm for $Q'$. Now it is straightforward to show that the algorithm $\mathbb{O} := \mathbb{O}' \circ \mathbb{S}$ that on input $x$ first computes $\mathbb{S}(x)$ and then simulates $\mathbb{O}'$ on $\mathbb{S}(x)$ is (strongly) almost optimal algorithm for $Q$. □

We come to the main result of this section.

THEOREM 3.3.

*(a)* TAUT *has an almost optimal algorithm* $\iff$ $p$-HALT$_>$ $\in \mathrm{XP}_{\mathrm{uni}}$.
*(b)* TAUT *has a strongly almost optimal algorithm* $\iff$ $p$-HALT$_>$ $\in \mathrm{FPT}_{\mathrm{uni}}$.

*Proof:* We first prove the directions from left to right in the equivalences. The classical problem HALT$_>$ underlying $p$-HALT$_>$ is easily seen to be coNP-complete. As TAUT has padding, for part (a), by the previous lemma, we may assume that HALT$_>$ has an almost optimal algorithm $\mathbb{O}$. Let $\mathbb{B}$ be a "brute force" algorithm that on input $\mathbb{M}$, a nondeterministic Turing machine, by systematically going through all runs of $\mathbb{M}$ on input $\lambda$ (the empty string) of length 1, of length 2,... computes $t_{\mathbb{M}}(\lambda)$, the least $k$ such that there is an accepting run of $\mathbb{M}$ on $\lambda$ of length $k$. If $\mathbb{M}$ has no such run, then $\mathbb{B}$ on input $\mathbb{M}$ does not halt.

We show that the following algorithm $\mathbb{O}^*$ simulating $\mathbb{B}$ and $\mathbb{O}$ in parallel witnesses that $p$-HALT$_>$ $\in$ XP$_{\text{uni}}$:

$\mathbb{O}^*$ // $\mathbb{M}$ a nondeterministic Turing machine, $1^n$ with $n \in \mathbb{N}$

1. in parallel simulate $\mathbb{B}$ on $\mathbb{M}$ and $\mathbb{O}$ on $\langle \mathbb{M}, 1^n \rangle$
2. **if** $\mathbb{O}$ halts first **then** answer accordingly
3. **if** $\mathbb{B}$ halts first **then**
4. **if** $n < t_{\mathbb{M}}(\lambda)$ **then** accept **else** reject.

Clearly, $\mathbb{O}^*$ decides $p$-HALT$_>$. We still have to verify that for fixed nondeterministic Turing machine $\mathbb{M}_0$ the algorithm $\mathbb{O}^*$ on input $\langle \mathbb{M}_0, 1^n \rangle$ runs in time polynomial in $n$.

*Case "*$\langle \mathbb{M}_0, 1^\ell \rangle \notin p$-HALT$_>$ *for some* $\ell \in \mathbb{N}$*"*: Then $\mathbb{B}$ will halt on input $\mathbb{M}_0$. Thus, in the worst case, $\mathbb{O}^*$ on input $\langle \mathbb{M}_0, 1^n \rangle$ has to wait till the simulation of $\mathbb{B}$ on $\mathbb{M}_0$ halts and then $\mathbb{O}^*$ must check whether the output $t_{\mathbb{M}_0}(\lambda)$ of the computation of $\mathbb{B}$ is bigger than $n$ or not and must answer accordingly. So in the worst case $\mathbb{O}^*$ takes time $O(t_{\mathbb{B}}(\mathbb{M}_0) + n)$.

*Case "*$\langle \mathbb{M}_0, 1^\ell \rangle \in p$-HALT$_>$ *for all* $\ell \in \mathbb{N}$*"*: In this case $\mathbb{B}$ on input $\mathbb{M}_0$ does not halt. Hence, the running time of $\mathbb{O}^*$ on input $\langle \mathbb{M}_0, 1^n \rangle$ is determined by that of $\mathbb{O}$ on this input. Here we will argue with the almost optimality of $\mathbb{O}$ and for this purpose we consider the algorithm $\mathbb{A}(\mathbb{M}_0)$ that on input $\langle \mathbb{M}, 1^n \rangle$ accepts if $\mathbb{M} = \mathbb{M}_0$; if $\mathbb{M} \neq \mathbb{M}_0$, it simulates $\mathbb{O}$ on input $\langle \mathbb{M}, 1^n \rangle$ and answers accordingly. Clearly, $\mathbb{A}(\mathbb{M}_0)$ decides HALT$_>$ and

$$t_{\mathbb{A}(\mathbb{M}_0)}\big( \langle \mathbb{M}_0, 1^n \rangle \big) \leq O(|\mathbb{M}_0| + n) \tag{8}$$

for all $n \in \mathbb{N}$. By the almost optimality of $\mathbb{O}$ there is a polynomial $p \in \mathbb{N}[X]$ (depending on $\mathbb{A}(\mathbb{M}_0)$ and hence on $\mathbb{M}_0$) such that for all $n$

$$t_{\mathbb{O}}\big( \langle \mathbb{M}_0, 1^n \rangle \big) \leq p\Big( t_{\mathbb{A}(\mathbb{M}_0)}(\langle \mathbb{M}_0, 1^n \rangle) + |\mathbb{M}_0| + n \Big). \tag{9}$$

Therefore, by (8), the algorithm $\mathbb{O}$ runs in time polynomial in $n$ on inputs of the form $\langle \mathbb{M}_0, 1^n \rangle$.

If the algorithm $\mathbb{O}$ is strongly almost optimal, there is a $d \in \mathbb{N}$ such that in the previous argument for all nondeterministic Turing machines $\mathbb{M}_0$ the polynomial $p$ can be chosen in $\mathbb{N}_d[X]$. Then, (8) and (9) show that

$$t_{\mathbb{O}}\big( \langle \mathbb{M}_0, 1^n \rangle \big) \leq f(|\mathbb{M}_0|) \cdot n^d$$

for all instances $\langle \mathbb{M}_0, 1^n \rangle$ of $p$-HALT$_>$ and some function $f : \mathbb{N} \to \mathbb{N}$, that is, $p$-HALT$_>$ $\in$ FPT$_{\text{uni}}$.

We turn to a proof of the implication from right to left in (b) (that of (a) is obtained by the obvious modifications). Assume that $\mathbb{B}$ is a Turing machine deciding whether $\langle \mathbb{M}, 1^n \rangle \in p$-HALT$_>$ in time

$$f(|\mathbb{M}|) \cdot n^d \tag{10}$$

for some $d \in \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$. The idea underlying the strongly almost optimal algorithm $\mathbb{O}$ for TAUT we aim at is simple: Using an effective enumeration of all Turing machines, for a given propositional formula $\alpha$ as input the algorithm $\mathbb{O}$ in a diagonal fashion performs steps of the machines in the enumeration on input $\alpha$; if $\mathbb{A}$, one of these machines, accepts $\alpha$, then $\mathbb{O}$ using the algorithm $\mathbb{B}$ checks whether (essentially) all inputs accepted by $\mathbb{A}$ in $\leq t_{\mathbb{A}}(\alpha)$ steps are tautologies; if so, $\mathbb{O}$ accepts $\alpha$.

We come to the details. For a deterministic Turing machine $\mathbb{A}$ we introduce machines $\mathbb{A}'$ and $\mathbb{A}''$, the first one being nondeterministic and the second deterministic. For the machine $\mathbb{A}'$ and a suitable quadratic polynomial $p_0 \in \mathbb{N}_2[X]$ we have:

(i) $L(\mathbb{A}) \subseteq \textsc{Taut} \iff t_{\mathbb{A}'}(\lambda) = \infty$;
(ii) If $\mathbb{A}$ accepts a string $x$ which is not a tautology, then $t_{\mathbb{A}'}(\lambda) \leq t_{\mathbb{A}}(x) + p_0(|x|)$.

```
𝔸′
 1. guess a string x ∈ Σ*
 2. simulate 𝔸 on input x
 3. if 𝔸 accepts then
 4.         if x is not a propositional formula then accept else
 5.                 guess a valuation v for x
 6.                 if v does not satisfy x then accept.
```

For the algorithm $\mathbb{A}''$ and every $x \in \Sigma^*$ we have:

(iii) $\mathbb{A}''$ accepts $x \iff \big(\mathbb{A}$ accepts $x$ and $t_{\mathbb{A}'}(\lambda) > t_{\mathbb{A}}(x) + p_0(|x|)\big)$.

```
𝔸″   // x ∈ Σ*
 1. simulate 𝔸 on input x thereby counting the number t_𝔸(x) of steps
 2. if 𝔸 rejects then reject
 3. u ← t_𝔸(x) + p_0(|x|)
 4. simulate 𝔹 on input ⟨𝔸′, 1^u⟩
                    (it is checked whether ⟨𝔸′, 1^u⟩ ∈ p-HALT_>)
 5.         if 𝔹 accepts then accept else reject.
```

Thus, we have:

(iv) $L(A'') \subseteq \textsc{Taut}$ $\big($by (ii) and (iii)$\big)$;
(v) if $L(A) \subseteq \textsc{Taut}$, then $L(A'') = L(A)$ $\big($by (i) and (iii)$\big)$.

If $\mathbb{A}''$ accepts $x$, then

$$\begin{aligned} t_{\mathbb{A}''}(x) &\leq O\Big(t_{\mathbb{A}}(x) + |x|^2 + t_{\mathbb{B}}(\langle \mathbb{A}', t_{\mathbb{A}}(x) + p_0(|x|)\rangle)\Big) \\ &\leq O\Big(t_{\mathbb{A}}(x) + |x|^2 + f(|\mathbb{A}'|) \cdot (t_{\mathbb{A}}(x) + p_0(|x|))^d\Big) \qquad \big(\text{by (10)}\big). \end{aligned}$$

As $p_0 \in \mathbb{N}_2[X]$, there exists a $p \in \mathbb{N}_{2d}[X]$ (depending on $\mathbb{A}$) such that

$$t_{\mathbb{A}''}(x) \leq p(t_{\mathbb{A}}(x) + |x|). \tag{11}$$

Let $\mathbb{T}$ be any algorithm deciding $\textsc{Taut}$ and $\mathbb{L}$ be an algorithm listing all Turing machines, that is, the algorithm $\mathbb{L}$, once having been started, eventually prints out exactly all Turing machines. For $i \in \mathbb{N}$ we denote by $\mathbb{A}_i$ the last machine printed out by $\mathbb{L}$ in $i$ steps; in particular, $\mathbb{A}_i$ is undefined if $\mathbb{L}$ hasn't printed any algorithm in $i$ steps. We define an algorithm $\mathbb{O}$ deciding $\textsc{Taut}$:

```
ℕ   // x ∈ Σ*
1.  ℓ ← 1
2.  simulate the ℓth step of 𝕋 on input x
3.  if 𝕋 halts then answer accordingly
4.  simulate the ℓth step of 𝕃
5.  for i = 0 to ℓ
6.          if 𝔸_i is defined then simulate the (ℓ − i)th step of 𝔸''_i on x
7.                  if 𝔸''_i accepts then accept
8.  ℓ ← ℓ + 1
9.  goto 2.
```

By (iv), it should be clear that $\mathbb{O}$ decides TAUT. We show that $\mathbb{O}$ is strongly almost optimal. Let $\mathbb{A}$ be any Turing machine deciding TAUT. We choose $i_0$ such that $\mathbb{A} = \mathbb{A}_{i_0}$. By (v), the algorithm $\mathbb{A}''_{i_0}$ decides TAUT, too. Therefore, $\mathbb{O}$ on input $\alpha \in$ TAUT accepts $\alpha$ in Line 7 for $\ell := i_0 + t_{\mathbb{A}''_{i_0}}(x)$ if it was not already accepted earlier. Thus,

$$t_{\mathbb{O}}(\alpha) = \big(i_0 + t_{\mathbb{A}''_{i_0}}(\alpha)\big)^{O(1)}$$

where the constant hidden in $O(1)$ does not depend on the algorithm $\mathbb{A}$. Thus by (11), there exists a constant $d' \in \mathbb{N}$ such that for all Turing machines $\mathbb{A}$ deciding TAUT there is a $p' \in \mathbb{N}_{d'}[X]$ such that for $\alpha \in$ TAUT

$$t_{\mathbb{O}}(\alpha) \le p'\big(t_{\mathbb{A}}(\alpha) + |\alpha|\big).$$

This shows that $\mathbb{O}$ is strongly almost optimal. □

REMARK 3.4. *For later reference we remark that in the proof of the implication from right to left we used the time bound* (10) *of the algorithm* $\mathbb{B}$ *only for inputs of the form* $\langle \mathbb{A}'_{i_0}, \ldots \rangle$. *As* $L(\mathbb{A}_{i_0}) =$ TAUT*, by (i) we have* $t_{\mathbb{A}'_{i_0}}(\lambda) = \infty$.

### 3.1. Variants of Theorem 3.3

There exist various variants and extensions of Theorem 3.3. In this subsection we derive a nondeterministic version, a space version, and an effective one. We leave it to the reader to combine the variants in order to get further insights.

*The nondeterministic variant.* The notion of almost optimal nondeterministic algorithm was introduced by Sadowski in [Sadowski 2002]; we recall it, thereby introducing a strong version, too.

DEFINITION 3.5. *A nondeterministic algorithm* $\mathbb{O}$ *accepting a problem* $Q \subseteq \Sigma^*$ *is* almost optimal *if for every nondeterministic algorithm* $\mathbb{A}$ *accepting* $Q$ *there is a polynomial* $p \in \mathbb{N}[X]$ *such that*

$$t_{\mathbb{O}}(x) \le p\big(t_{\mathbb{A}}(x) + |x|\big)$$

*for all* $x \in Q$. *If there is a* $d \in \mathbb{N}$ *such that for all* $\mathbb{A}$ *the polynomial* $p$ *can be chosen in* $\mathbb{N}_d[X]$, *then* $\mathbb{O}$ *is* strongly almost optimal.

Recall that para-$\mathrm{NP}_{\mathrm{uni}}$ is the class of parameterized problems $(Q, \kappa)$ accepted by a nondeterministic algorithm $\mathbb{A}$ such that $t_{\mathbb{A}}(x) \le f(\kappa(x)) \cdot |x|^{O(1)}$ for all $x \in Q$ and some function $f : \mathbb{N} \to \mathbb{N}$. The following nondeterministic version of Theorem 3.3 is proven in the same way.

THEOREM 3.6.

*(a)* TAUT *has an almost optimal nondeterministic algorithm* $\iff$ $p$-HALT$_>$ $\in$ XNP$_{\text{uni}}$.
*(b)* TAUT *has a strongly almost optimal nondeterministic algorithm* $\iff$ $p$-HALT$_>$ $\in$ para-NP$_{\text{uni}}$.

As XP$_{\text{uni}} \subseteq$ XNP$_{\text{uni}}$ and FPT$_{\text{uni}} \subseteq$ para-NP$_{\text{uni}}$ we get from Theorem 3.3 and Theorem 3.6 the following corollary whose part (a) was already proven in [Sadowski 2002]:

COROLLARY 3.7.

*(a) If* TAUT *has an almost optimal algorithm, then it also has an almost optimal nondeterministic algorithm.*
*(b) If* TAUT *has a strongly almost optimal algorithm, then it also has a strongly almost optimal nondeterministic algorithm.*

*The space variant.* Recall that $s_{\mathbb{A}}(x)$ denotes the minimum space used by an accepting run of the nondeterministic algorithm $\mathbb{A}$ on input $x$.

DEFINITION 3.8. *A deterministic (nondeterministic) algorithm $\mathbb{O}$ deciding $Q$ is* almost space optimal *for $Q$ if for every deterministic (nondeterministic) algorithm $\mathbb{A}$ which decides (accepts) $Q$ there is a $d \in \mathbb{N}$ such that for all $x \in Q$*

$$s_{\mathbb{O}}(x) \leq d \cdot \big(s_{\mathbb{A}}(x) + \log |x|\big).$$

We say that an algorithm $\mathbb{A}$ is *loop-free* if $t_{\mathbb{A}}(x) < \infty$ implies $s_{\mathbb{A}}(x) < \infty$ for all $x \in \Sigma^*$. For an algorithm $\mathbb{C}$ the loop-free algorithm $\mathbb{C}^+$, claimed to exist in the following well-known lemma, on input $x$ simulates $\mathbb{C}$ on $x$ and thereby records the space and the time $\mathbb{C}$ uses; once the time exceeds the number of configurations using the corresponding space, $\mathbb{C}^+$ rejects.

LEMMA 3.9. *One can effectively assign to every algorithm $\mathbb{C}$ a loop-free algorithm $\mathbb{C}^+$ such that:*

– $L(\mathbb{C}) = L(\mathbb{C}^+)$*;*
– *for all* $x \in \Sigma^*$*,* $(s_{\mathbb{C}}(x) < \infty \iff s_{\mathbb{C}^+}(x) < \infty)$*;*
– *for all* $x \in \Sigma^*$ *with* $s_{\mathbb{C}}(x) < \infty$ *we have* $s_{\mathbb{C}}(x) \leq s_{\mathbb{C}^+}(x) \leq O(s_{\mathbb{C}}(x) + \log |x|)$.

THEOREM 3.10.

*(a)* TAUT *has an almost space optimal algorithm* $\iff$ $p$-HALT$_>$ $\in$ XL$_{\text{uni}}$.
*(b)* TAUT *has an almost space optimal nondeterministic algorithm* $\iff$ $p$-HALT$_>$ $\in$ XNL$_{\text{uni}}$.

*Proof:* Note first that the space variant of Lemma 3.2 holds if we assume that $Q$ is logspace reducible to $Q'$. Then one shows the direction from left to right of the statements (a) and (b) along the lines of the corresponding proof of Theorem 3.3.

We sketch the main changes needed for the implications from right to left. For a Turing machine $\mathbb{A}$ we define $\mathbb{A}'$ and $\mathbb{A}''$ as there. For part (a), again let $\mathbb{T}$ be an algorithm deciding TAUT. We choose a listing $\mathbb{A}_1, \mathbb{A}_2, \ldots$ of all Turing machines and let $\mathbb{L}$ be the listing $\mathbb{A}_1^+, \mathbb{A}_2^+, \ldots$ of the loop-free Turing machines obtained from it by the previous lemma. The algorithm $\mathbb{O}$ deciding TAUT is defined by:

```
ℕ   // x ∈ Σ*
1.  ℓ ← 1
2.  simulate the ℓth step of 𝕋 on input x
3.  if 𝕋 halts then answer accordingly
4.  simulate 𝕃 using space at most ℓ · log |x|
5.        if 𝕃 prints a machine 𝔸 then simulate 𝔸″ on x using
                                          space at most ℓ · log |x|
6.              if 𝔸″ accepts then accept
7.  ℓ ← ℓ + 1
8.  goto 2.
```

Using a listing of *loop-free* machines, ensures that the simulation of $\mathbb{A}''$ in Line 5 eventually stops as $\mathbb{A}''$ eventually will exceed space $\ell \cdot \log |x|$. One then shows that $\mathbb{O}$ is almost space optimal.

For part (b) one uses a listing $\mathbb{L}$ of all nondeterministic Turing machines and defines the nondeterministic algorithm $\mathbb{O}$ deciding TAUT by:

```
ℕ   // x ∈ Σ*
1.  guess i ≥ 1
2.  simulate 𝕃 till it outputs the ith machine, say, 𝔸
3.  simulate 𝔸″ on x
4.  if 𝔸″ accepts then accept.
```

□

As $\text{XL}_{\text{uni}} \subseteq \text{XP}_{\text{uni}}$ and $\text{XNL}_{\text{uni}} \subseteq \text{XNP}_{\text{uni}}$, we obtain from the previous result the following corollary by Theorem 3.3 (a) and Theorem 3.6 (a).

COROLLARY 3.11.

*(a) If* TAUT *has an almost space optimal algorithm, then it has an almost (time) optimal algorithm.*
*(b) If* TAUT *has an almost space optimal nondeterministic algorithm, then it has an almost (time) optimal nondeterministic algorithm.*

*The effective variant.* In [Chen and Flum 2010a] we have shown:

PROPOSITION 3.12.

*(1) If* $\text{NP}[\text{TC}] \neq \text{P}[\text{TC}]$*, then* $p$-$\text{HALT}_{>} \notin \text{FPT}$.
*(2) If* $\text{NP}[\text{TC}] \not\subseteq \text{P}[\text{TC}^{\log \text{TC}}]$*, then* $p$-$\text{HALT}_{>} \notin \text{XP}$.

Here $\text{NP}[\text{TC}] \neq \text{P}[\text{TC}]$ means that for all *time constructible* and increasing functions $h$ the class of problems decidable in *nondeterministic polynomial time* in $h$ and the class of problems decidable in *deterministic polynomial time* in $h$ are distinct, that is, $\text{NDTIME}(h^{O(1)}) \neq \text{DTIME}(h^{O(1)})$. By taking as $h$ the identity function on $\mathbb{N}$ and the function $n \mapsto 2^n$ we see that $\text{NP}[\text{TC}] \neq \text{P}[\text{TC}]$ implies NP ≠ P and NE ≠ E, respectively. And $\text{NP}[\text{TC}] \not\subseteq \text{P}[\text{TC}^{\log \text{TC}}]$ means that $\text{NTIME}(h^{O(1)}) \not\subseteq \text{DTIME}(h^{O(\log h)})$ for every time constructible and increasing function $h$. In [Chen and Flum 2010d] we have related the assumptions $\text{NP}[\text{TC}] \neq \text{P}[\text{TC}]$ and $\text{NP}[\text{TC}] \not\subseteq \text{P}[\text{TC}^{\log \text{TC}}]$ to the so-called *Measure Hypothesis* [Hitchcock and Pavan 2004].

What is the *natural* effective version of almost optimal algorithm for TAUT and is it equivalent to the statement $p$-HALT$_>$ $\in$ XP? If in the definition of almost optimal algorithm (Definition 3.1) the polynomial $p$ in (7) can be computed from $\mathbb{A}$, then the algorithm $\mathbb{A}$ is said to be effectively almost optimal. More precisely:

DEFINITION 3.13. *An algorithm $\mathbb{O}$ deciding $Q$ is* effectively (and strongly) almost optimal *if there is a computable function $p : \Sigma^* \to \mathbb{N}[X]$ such that for every deterministic algorithm $\mathbb{A}$ deciding $Q$*

$$t_{\mathbb{O}}(x) \leq p(\mathbb{A})\big(t_{\mathbb{B}}(x) + |x|\big)$$

*for all $x \in Q$ (and there is a $d \in \mathbb{N}$ such that even $p : \Sigma^* \to \mathbb{N}_d[X]$).*

We could not show that the existence of an effectively almost optimal algorithm for TAUT is equivalent to $p$-HALT$_>$ $\in$ XP; however by analyzing the proof of Theorem 3.3, we isolate a property of $p$-HALT$_>$ $\in$ XP equivalent to the existence of such an algorithm for TAUT.

We say that $p$-HALT$_>$ is FPT-*decidable* (XP-*decidable*) *on non-halting machines* if there is an algorithm $\mathbb{A}$ deciding $p$-HALT$_>$, a computable function $f : \mathbb{N} \to \mathbb{N}$, and $c \in \mathbb{N}$ such that for all nondeterministic Turing machines $\mathbb{M}$ with $t_{\mathbb{M}}(\lambda) = \infty$ we have for all $n \in \mathbb{N}$

$$t_{\mathbb{A}}(\langle \mathbb{M}, n \rangle) \leq f(|\mathbb{M}|) \cdot n^c \qquad \left(t_{\mathbb{A}}(\langle \mathbb{M}, n \rangle) \leq n^{f(|\mathbb{M}|)}\right).$$

LEMMA 3.14.

*(a) If $p$-HALT$_>$ $\in$ FPT ($p$-HALT$_>$ $\in$ XP), then $p$-HALT$_>$ is* FPT-*decidable* (XP-*decidable*) *on non-halting machines.*
*(b) If $p$-HALT$_>$ is* FPT-*decidable* (XP-*decidable*) *on non-halting machines, then $p$-HALT$_>$ $\in$ FPT$_{\text{uni}}$ ($p$-HALT$_>$ $\in$ XP$_{\text{uni}}$).*

*Proof:* Part (a) is clear by the definitions. For part (b) we combine an algorithm witnessing that $p$-HALT$_>$ is FPT-decidable (XP-decidable) on non-halting machines, with an algorithm that by "brute-force" on input $\langle \mathbb{M}, 1^n \rangle$ computes $t_{\mathbb{M}}(\lambda)$. □

The effective variant of Theorem 3.3 reads as follows:

PROPOSITION 3.15.

*(a) TAUT has an effectively almost optimal algorithm $\iff$ $p$-HALT$_>$ is* XP-*decidable on non-halting machines.*
*(b) TAUT has an effectively and strongly almost optimal algorithm $\iff$ $p$-HALT$_>$ is* FPT-*decidable on non-halting machines.*

*Proof:* The equivalences are obtained as that of Theorem 3.3; for the implication from right to left we use Remark 3.4. □

COROLLARY 3.16.

*(1) If* NP[TC] $\neq$ P[TC]*, then* TAUT *has no effectively and strongly almost optimal algorithm.*
*(2) If* NP[TC] $\not\subseteq$ P[TC$^{\log \text{TC}}$]*, then* TAUT *has no effectively almost optimal algorithm.*

*Proof:* A slight modification of the proof of Proposition 3.12 in [Chen and Flum 2010a] yields that $p$-HALT$_>$ is not FPT-decidable (XP-decidable) on non-halting machines if NP[TC] $\neq$ P[TC] (NP[TC] $\not\subseteq$ P[TC$^{\log \text{TC}}$]) holds. Now the claims follow from Proposition 3.15. □

On the other hand we have:

COROLLARY 3.17. *If* NE = E*, then* TAUT *has an effectively and strongly almost optimal algorithm.*

*Proof:* We consider the variant of $\text{HALT}_>$, where instead of the string $1^n$ we have $n$ in binary as part of the input. Of course, this problem is in NE and hence by assumption, it is solvable by a deterministic machine in time $2^{c\cdot(\log n+|\mathbb{M}|)}$ for some $c \in \mathbb{N}$. Thus, $p$-$\text{HALT}_> \in$ FPT and therefore the claim follows from Proposition 3.15 by Lemma 3.14. □

## 4. INVARIANT LOGICS AND $P$-HALT$_>$.

For $\text{C} \in \{\text{L}, \text{NL}, \text{P}\}$ we introduce a logic $L(\text{C})_{\text{inv}}$. We state the main result of this section, even though we haven't introduced some of the concepts so far.

THEOREM 4.1. *Let* $\text{C} \in \{\text{L}, \text{NL}, \text{P}\}$*,* $\text{C}' \in \{\text{L}, \text{NL}, \text{P}, \text{NP}\}$*, and* $\text{C} \subseteq \text{C}'$*. Then*

$$L(\text{C})_{\text{inv}} \textit{ is a } \text{C}'\textit{-bounded logic for } \text{C} \iff p\text{-}\text{HALT}_> \in \text{XC}'_{\text{uni}}.$$

We start by recalling the concepts from logic we need.

*Structures.* A *vocabulary* $\tau$ is a finite set of relation symbols. Each relation symbol has an *arity*. A *structure* $\mathcal{A}$ of vocabulary $\tau$, or $\tau$*-structure* (or, simply structure), consists of a nonempty set $A$ called the *universe*, and an interpretation $R^{\mathcal{A}} \subseteq A^r$ of each $r$-ary relation symbol $R \in \tau$. *All structures in this paper are assumed to have finite universe.* Where necessary we identify structures with strings in $\Sigma^*$ in a natural way.

*Logics capturing complexity classes.* For our purposes a *logic* $L$ consists

- for every vocabulary $\tau$ of a set $L[\tau]$ of strings, the set of $L$*-sentences of vocabulary* $\tau$, and of an algorithm that for every $\tau$ and every string $\xi$ decides whether $\xi \in L[\tau]$ (in particular, $L[\tau]$ is decidable for every $\tau$);
- of a *satisfaction relation* $\models_L$; if $(\mathcal{A}, \varphi) \in \models_L$ (written: $\mathcal{A} \models_L \varphi$), then $\mathcal{A}$ is a $\tau$-structure and $\varphi \in L[\tau]$ for some vocabulary $\tau$; furthermore for each $\varphi \in L[\tau]$ the class

$$\text{MOD}_L(\varphi) := \{\mathcal{A} \mid \mathcal{A} \models_L \varphi\}$$

of *models of* $\varphi$ is closed under isomorphism.

From now on, if we say "let $\varphi$ be an $L$-sentence," we mean that, in addition to $\varphi$, a vocabulary $\tau$ with $\varphi \in L[\tau]$ is given.

Recall that C and C′ range over the complexity classes L, P, NL, and NP.

DEFINITION 4.2. *Let* $L$ *be a logic and* C *a complexity class.*

*(a)* $L$ is a logic for C *if for all vocabularies* $\tau$ *and all classes* $S$ *of* $\tau$*-structures closed under isomorphism we have*

$$S \in \text{C} \iff S = \text{MOD}_L(\varphi) \textit{ for some } \varphi \in L[\tau].$$

*(b) Let* C′ *be a deterministic (nondeterministic) complexity class.* $L$ is a C′-bounded logic for C *if* $L$ *is a logic for* C *and if there is a deterministic (nondeterministic) algorithm* $\mathbb{A}$ *deciding (accepting)* $\models_L$*, which for fixed* $\varphi$ *witnesses that* $\text{MOD}_L(\varphi) \in \text{C}'$.

Clearly, if $L$ is a C′-bounded logic for C, then $\text{C} \subseteq \text{C}'$. If $\text{C}' = \text{C}$, then property (b) yields the implication from right to left in part (a). One expects from a logic $L$ capturing the complexity class C that it is C′-bounded for C with $\text{C}' = \text{C}$. In fact, one expects that $L$ can be viewed as a higher programming language for C, that is, that for fixed $L$-sentence $\varphi$ the algorithm $\mathbb{A}$ in (b) witnesses that $\text{MOD}_L(\varphi) \in \text{C}$. However we use

this more liberal, a little bit artificial notion as in this way we obtain some nontrivial consequences from our results. Fagin [Fagin 1974] has shown that there exist NP-bounded logics for NP (e.g., the logic consisting of the $\Sigma^1_1$-sentences of second-order logic), while for $\mathrm{C} \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$ it is open whether there is a C-bounded logic for C.

We introduce the notion of an effectively $\mathrm{C}'$-bounded logic for C only for $\mathrm{C}' = \mathrm{C} = \mathrm{P}$, as we do not use it for other complexity classes. If $L$ is a P-bounded logic for P, then for every $L$-sentence $\varphi$ the algorithm $\mathbb{A}$ of Definition 4.2 (b) witnesses that $\mathrm{MOD}_L(\varphi) \in \mathrm{P}$. However, we do not necessarily know ahead of time the bounding polynomial. The logic $L$ *is an effectively* P*-bounded logic for* P if $L$ is a P-bounded logic for P and if in addition to the algorithm $\mathbb{A}$ as in Definition 4.2 (b) there is a computable function that assigns to every $L$-sentence $\varphi$ a polynomial $q \in \mathbb{N}[X]$ such that $\mathbb{A}$ decides whether $\mathcal{A} \models_L \varphi$ in $\leq q(|\mathcal{A}|)$ steps.

For every vocabulary $\tau$ we let $\tau_< := \tau \cup \{<\}$, where $<$ is a binary relation symbol not in $\tau$ chosen in some canonical way. A $\tau_<$-structure $\mathcal{A}$ is *ordered* if $<^{\mathcal{A}}$ is a (total and linear) ordering of the universe $A$ of $\mathcal{A}$.

We say that a logic $L$ is a $\mathrm{C}'$*-bounded logic for* C *on ordered structures* if (a) and (b) of Definition 4.2 hold for *ordered* structures and classes of *ordered* structures. In (b), for fixed $\varphi \in L[\tau_<]$ the algorithm $\mathbb{A}$ must witness that the class of ordered models of $\varphi$ is in $\mathrm{C}'$. It should be clear what we mean by an *effectively* P*-bounded logic for* P *on ordered structures*.

We denote by FO, DTC, TC, and LFP *first-order logic, deterministic transitive closure logic, transitive closure logic*, and *least fixed-point logic*, respectively. We assume familiarity with FO. Concerning DTC, TC, and LFP, essentially we only need the following properties:

THEOREM 4.3. [Immerman 1986; 1987; 1988; Vardi 1982] *On ordered structures,* DTC *is an* L*-bounded logic for* L*,* TC *is an* NL*-bounded logic for* NL*, and* LFP *is an effectively* P*-bounded logic for* P.

For $\mathrm{C} = \mathrm{L}$, $\mathrm{C} = \mathrm{NL}$, and $\mathrm{C} = \mathrm{P}$ we let $L(\mathrm{C})$ be the logic DTC, TC, and LFP, respectively.

*Invariant sentences and the logic* $L_{\mathrm{inv}}$. We start by introducing the notion of (order-) invariance.

DEFINITION 4.4. *Let $L$ be a logic.*

*(1) Let $\varphi$ be an $L[\tau_<]$-sentence and $n \in \mathbb{N}$. We say that $\varphi$ is* $\leq n$-invariant *if for all $\tau$-structures $\mathcal{A}$ with $|A| \leq n$ and all orderings $<_1$ and $<_2$ on $A$ we have*

$$(\mathcal{A}, <_1) \models_L \varphi \iff (\mathcal{A}, <_2) \models_L \varphi.$$

*(2) The* parameterized $L$-invariant problem *is the problem*

| $p$-$L$-INV | |
|---|---|
| *Instance:* | An $L$-sentence $\varphi$ and $1^n$ with $n \in \mathbb{N}$. |
| *Parameter:* | $\lvert\varphi\rvert$. |
| *Question:* | Is $\varphi \leq n$-invariant? |

*For better readability we often will write $\langle \varphi, n \rangle \in p$-$L$-INV instead of $\langle \varphi, 1^n \rangle \in p$-$L$-INV.*

We define the logic $L_{\mathrm{inv}}$. For every vocabulary $\tau$ we set

$$L_{\mathrm{inv}}[\tau] := L[\tau_<],$$

and we define the satisfaction relation by

$$\mathcal{A} \models_{L_{\text{inv}}} \varphi \iff \Big(\langle \varphi, |A| \rangle \in p\text{-}L\text{-}\textsc{Inv} \text{ and } (\mathcal{A}, <) \models_L \varphi \text{ for some ordering} < \text{ on } A\Big). \quad (12)$$

Assume that for every $L$-sentence $\varphi$, say, of vocabulary $\tau$ the string $\neg\varphi$ is an $L[\tau]$-sentence such that $\textsc{Mod}(\neg\varphi) = \big\{\mathcal{A} \;\big|\; \mathcal{A} \text{ a } \tau\text{-structure and } \mathcal{A} \notin \textsc{Mod}(\varphi)\big\}$. As $\langle \varphi, n\rangle \in p\text{-}L\text{-}\textsc{Inv}$ if and only if $\langle \neg\varphi, n\rangle \in p\text{-}L\text{-}\textsc{Inv}$, we get for every structure $\mathcal{A}$

$$\langle \varphi, |A| \rangle \in p\text{-}L\text{-}\textsc{Inv} \iff \big(\mathcal{A} \models_{L_{\text{inv}}} \varphi \text{ or } \mathcal{A} \models_{L_{\text{inv}}} \neg\varphi\big). \quad (13)$$

and thus for $n \geq 1$,

$$\langle \varphi, n\rangle \in p\text{-}L\text{-}\textsc{Inv} \iff \big(\mathcal{A}(\tau, n) \models_{L_{\text{inv}}} \varphi \;\text{ or }\; \mathcal{A}(\tau, n) \models_{L_{\text{inv}}} \neg\varphi\big), \quad (14)$$

where $\mathcal{A}(\tau, n)$ is the $\tau$-structure with universe $\{1, \dots, n\}$ and empty relations (that is, every relation symbol in $\tau$ is interpreted by the empty set).

Now that we have introduced all concepts needed to understand the statement of Theorem 4.1, we start with a series of lemmas that finally will yield a proof of it.

LEMMA 4.5. *If* $\mathrm{C} \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$*, then* $L(\mathrm{C})_{\text{inv}}$ *is a logic for* C.

*Proof:* For a class $S$ of $\tau$-structures let $S_<$ be the following class of $\tau_<$-structures:

$$S_< := \big\{(\mathcal{A}, <) \;\big|\; \mathcal{A} \in S \text{ and } < \text{ is an ordering on } A\big\}.$$

Clearly, $\big(S \in \mathrm{C} \iff S_< \in \mathrm{C}\big)$. Now, if $S \in \mathrm{C}$, then, by Theorem 4.3, there is an $L(\mathrm{C})[\tau_<]$-sentence $\varphi$ such that $S_< = \textsc{Mod}_{L(\mathrm{C})}(\varphi)$. Hence, $S = \textsc{Mod}_{L(\mathrm{C})_{\text{inv}}}(\varphi)$.

Conversely, let $\varphi$ be an $L(\mathrm{C})_{\text{inv}}[\tau]$-sentence. If $\varphi$ is not $\leq n$-invariant for some $n \in \mathbb{N}$, then $\textsc{Mod}_{L(\mathrm{C})_{\text{inv}}}(\varphi)$ contains only structures with universes of less than $n$ elements and thus only finitely many up to isomorphism; hence it is in C. Otherwise, $\textsc{Mod}_{L(\mathrm{C})_{\text{inv}}}(\varphi)_< = \textsc{Mod}_{L(\mathrm{C})}(\varphi)$. As the latter class is in C (by Theorem 4.3), so is the class $\textsc{Mod}_{L(\mathrm{C})_{\text{inv}}}(\varphi)$. □

LEMMA 4.6. *Let* $\mathrm{C} \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$*,* $\mathrm{C}' \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}\}$*, and* $\mathrm{C} \subseteq \mathrm{C}'$*. Then* $L(\mathrm{C})_{\text{inv}}$ *is a* $\mathrm{C}'$*-bounded logic for* C *if and only if* $p\text{-}L(\mathrm{C})\text{-}\textsc{Inv} \in \mathrm{XC}'_{\text{uni}}$.

*Proof:* By the preceding lemma, we already know that $L(\mathrm{C})_{\text{inv}}$ is a logic for C, that is, part (a) of Definition 4.2 is fulfilled. By (12) and Theorem 4.3, part (b) of this definition is fulfilled if $p\text{-}L(\mathrm{C})\text{-}\textsc{Inv} \in \mathrm{XC}'_{\text{uni}}$. Conversely, if part (b) is fulfilled, then, by (14), $p\text{-}L(\mathrm{C})\text{-}\textsc{Inv} \in \mathrm{XC}'_{\text{uni}}$. □

In the next section we will prove:

LEMMA 4.7. *Let* $\mathrm{C} \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$ *and* $\mathrm{C}' \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}\}$*. Then*

$$\begin{aligned} p\text{-}\textsc{Halt}_> \in \mathrm{XC}'_{\text{uni}} &\iff p\text{-}L(\mathrm{C})\text{-}\textsc{Inv} \in \mathrm{XC}'_{\text{uni}} \\ &\iff p\text{-}\mathrm{FO}_{\text{inv}}\text{-}\textsc{Inv} \in \mathrm{XC}'_{\text{uni}}. \end{aligned}$$

*Proof of Theorem 4.1*: Let C and $\mathrm{C}'$ be as in the statement of Theorem 4.1. Then

$$\begin{aligned} L(\mathrm{C})_{\text{inv}} \text{ is a } \mathrm{C}'\text{-bounded logic for C} &\iff p\text{-}L(\mathrm{C})\text{-}\textsc{Inv} \in \mathrm{XC}'_{\text{uni}} \quad \text{(by Lemma 4.6)} \\ &\iff p\text{-}\textsc{Halt}_> \in \mathrm{XC}'_{\text{uni}} \quad \text{(by Lemma 4.7)}. \end{aligned}$$

□

We draw some consequences from Theorem 4.1. Even though it is not known whether the existence of an L-bounded logic for L implies the existence of a P-bounded logic for P, we get:

COROLLARY 4.8. *If* $\mathrm{DTC}_{\mathrm{inv}}$ *is an* L*-bounded logic for* L*, then* $\mathrm{TC}_{\mathrm{inv}}$ *is an* NL*-bounded logic for* NL *and* $\mathrm{LFP}_{\mathrm{inv}}$ *is a* P*-bounded logic for* P.

*Proof:* As $\mathrm{XL}_{\mathrm{uni}} \subseteq \mathrm{XNL}_{\mathrm{uni}}$ and $\mathrm{XL}_{\mathrm{uni}} \subseteq \mathrm{XP}_{\mathrm{uni}}$, the results follow from Theorem 4.1. □

As $\mathrm{XNL}_{\mathrm{uni}} \not\subseteq \mathrm{XP}_{\mathrm{uni}}$, so far we do not know whether $\mathrm{LFP}_{\mathrm{inv}}$ is a P-bounded logic for P if $\mathrm{TC}_{\mathrm{inv}}$ is an NL-bounded logic for NL. Nevertheless, in Corollary 5.3 of the next section we will see that this implication holds, too.

COROLLARY 4.9.

*(a) If there is an algorithm deciding* $\mathcal{A} \models_{\mathrm{FO}_{\mathrm{inv}}} \varphi$ *which, for fixed first-order* $\varphi$*, requires space logarithmic in* $|\mathcal{A}|$*, then* $\mathrm{DTC}_{\mathrm{inv}}$ *is an* L*-bounded logic for* L.
*(b) If* $\mathrm{DTC}_{\mathrm{inv}}$ *is a* P*-bounded logic for* L*, then* $\mathrm{LFP}_{\mathrm{inv}}$ *is a* P*-bounded logic for* P.

*Proof:* (a) Assume that the hypothesis on $\mathrm{FO}_{\mathrm{inv}}$ in (a) is fulfilled. Then, by (14), $p$-FO-INV $\in \mathrm{XL}_{\mathrm{uni}}$ and thus $p$-$\mathrm{HALT}_{>} \in \mathrm{XL}_{\mathrm{uni}}$ by Lemma 4.7. Now our claim follows from Theorem 4.1. The proof of (b) is similar even easier. □

It is not known whether the existence of a P-bounded logic for P implies that of an *effectively* P-bounded logic for P. However, we can show:

PROPOSITION 4.10. *If* $\mathrm{LFP}_{\mathrm{inv}}$ *is a* P*-bounded logic for* P*, then there is an effectively* P*-bounded logic for* P.

*Proof:* Assume that $\mathrm{LFP}_{\mathrm{inv}}$ is a P-bounded logic for P and let $\mathbb{A}$ be an algorithm according to Definition 4.2 (b). Let $\mathbb{B}$ be the algorithm that decides $\langle \varphi, n \rangle \in p$-$\mathrm{LFP}_{\mathrm{inv}}$-INV via the equivalence (14), where it checks the disjuncts on the right hand side of (14) by applying the algorithm $\mathbb{A}$. Then, for fixed $\varphi$ the algorithm $\mathbb{B}$ has running time polynomial in $n$. We define the logic $T(\mathrm{LFP}_{\mathrm{inv}})$, *time-clocked* $\mathrm{LFP}_{\mathrm{inv}}$, by:

– for every vocabulary $\tau$

$$T(\mathrm{LFP}_{\mathrm{inv}})[\tau] := \big\{ (\varphi, p) \;\big|\; \varphi \in \mathrm{LFP}_{\mathrm{inv}}[\tau] \text{ and } p \in \mathbb{N}[X] \big\};$$

– $\mathcal{A} \models_{T(\mathrm{LFP}_{\mathrm{inv}})} \varphi$ iff (a) and (b) hold where
  (a) $\mathbb{B}$ accepts $\langle \varphi, |A| \rangle$ in $\leq p(|A|)$ steps;
  (b) $(\mathcal{A}, <) \models_{\mathrm{LFP}} \varphi$ for some ordering $<$, say for the ordering of $A$ induced by (the string) $\mathcal{A}$.

It is not hard to verify that $T(\mathrm{LFP}_{\mathrm{inv}})$ is an effectively P-bounded logic for P (here we use for checking (b) an algorithm deciding $\models_{\mathrm{LFP}}$ that witnesses that LFP is an effectively P-bounded logic for P on ordered structures (see Theorem 4.3)). □

REMARK 4.11. *In a slightly different way but using the same idea one can define the time-clocked version* $T(L)$ *for any* P*-bounded logic* $L$ *for* P*. However, in general,* $T(L)$ *is not even a logic, as the class of models of a* $T(L)$*-sentence must not be closed under isomorphism. In the case of* $T(\mathrm{LFP}_{\mathrm{inv}})$ *this is guaranteed by the fact that condition (a) in the definition of* $\mathcal{A} \models_{T(\mathrm{LFP}_{\mathrm{inv}})} \varphi$ *only refers to the cardinality of the universe of* $\mathcal{A}$*.*

EXAMPLE 4.12. *There are* NP*-bounded logics for* P*. In fact, consider the logic* $\mathrm{LFP}_{\mathrm{inv}}(\mathrm{not})$ *which has the same syntax as* $\mathrm{LFP}_{\mathrm{inv}}$*, that is,*

$$\mathrm{LFP}_{\mathrm{inv}}(\mathrm{not})[\tau] := \mathrm{LFP}_{\mathrm{inv}}[\tau]$$

*and whose semantics is given by*

$$\mathcal{A} \models_{\mathrm{LFP}_{\mathrm{inv}}(\mathrm{not})} \varphi \iff \textit{not } \mathcal{A} \models_{\mathrm{LFP}_{\mathrm{inv}}} \varphi.$$

*As* P *is closed under complements,* $\mathrm{LFP}_{\mathrm{inv}}(\mathrm{not})$ *is a logic for* P. *Using the definition of the semantics, one easily verifies that* $\mathrm{LFP}_{\mathrm{inv}}(\mathrm{not})$ *is an* NP*-bounded logic for* P.

### 4.1. A variant

What does $p$-HALT$_>$ $\in$ XC mean for the logic $L(\mathrm{C})_{\mathrm{inv}}$? For simplicity, we only consider the case C = P (for which we already answered this question in [Chen and Flum 2010a]). Arguing as in the proof of Lemma 4.6 one gets

$$\mathrm{LFP}_{\mathrm{inv}} \text{ is an effectively P-bounded logic for P} \iff p\text{-LFP-INV} \in \mathrm{XP}.$$

Using this equivalence, one can obtain:

THEOREM 4.13. $\mathrm{LFP}_{\mathrm{inv}}$ *is an effectively* P*-bounded logic for* P $\iff$ $p$-HALT$_>$ $\in$ XP.

The result for $p$-HALT$_>$ $\in$ FPT is more involved and we refer the reader to [Chen and Flum 2010a] for the details. There for an LFP-formula $\varphi$ we introduce its *depth*, the maximum nesting depth of LFP-operators in $\varphi$, and its *width,* essentially the maximum of the number of variables in subformulas of $\varphi$. It is not hard to see that there is an algorithm deciding whether $\mathcal{A} \models_{\mathrm{LFP}} \varphi$ in time

$$|\varphi| \cdot |\mathcal{A}|^{O((1+\mathrm{depth}(\varphi))\cdot \mathrm{width}(\varphi))}.$$

We say that $\mathrm{LFP}_{\mathrm{inv}}$ is an *(effectively) depth-width* P*-bounded logic for* P if it is a logic for P and there is a (computable) function $h : \mathbb{N} \to \mathbb{N}$ and an algorithm $\mathbb{A}$ deciding whether $\mathcal{A} \models_{\mathrm{LFP}_{\mathrm{inv}}} \varphi$ in time

$$h(|\varphi|) \cdot |\mathcal{A}|^{O((1+\mathrm{depth}(\varphi))\cdot \mathrm{width}(\varphi))}.$$

The following holds:

THEOREM 4.14.

*(a)* $\mathrm{LFP}_{\mathrm{inv}}$ *is a depth-width* P*-bounded logic for* P $\iff$ $p$-HALT$_>$ $\in \mathrm{FPT}_{\mathrm{uni}}$.
*(b)* $\mathrm{LFP}_{\mathrm{inv}}$ *is an effectively depth-width* P*-bounded logic for* P $\iff$ $p$-HALT$_>$ $\in$ FPT.

## 5. SLICEWISE DOWNWARD MONOTONE PARAMETERIZED PROBLEMS

A parameterized problem $(Q, \kappa)$ is *slicewise downward monotone* if all elements of $Q$ have the form $\langle x, 1^n\rangle$ with $x \in \Sigma^*$ and $n \in \mathbb{N}$, if $\kappa(\langle x, 1^n\rangle) = |x|$, and finally if the slices are downward monotone, that is, for all $x \in \Sigma^*$ and $n, n' \in \mathbb{N}$

$$\langle x, 1^n\rangle \in Q \text{ and } n' < n \text{ imply } \left\langle x, 1^{n'}\right\rangle \in Q.$$

Hence, $p$-HALT$_>$ and $p$-$L$-INV (for any logic $L$) are slicewise downward monotone. As already done for $p$-$L$-INV, often for a slicewise downward monotone $(Q, \kappa)$ we will write $\langle x, n\rangle \in Q$ instead of $\langle x, 1^n\rangle \in Q$.

For any logic $L$ the following parameterized problem is downward monotone, too.

| $p$-$L$-MODEL$_>$ | |
|---|---|
| *Instance:* | An $L$-sentence $\varphi$ and $1^n$ with $n \in \mathbb{N}$. |
| *Parameter:* | $\lvert\varphi\rvert$. |
| *Question:* | Does the universe of every $L$-model of $\varphi$ have more than $n$ elements? |

In this section we aim to show that all slicewise downward monotone parameterized problems introduced so far have the same computational complexity.

We denote by $\mathrm{coXNL}_{\mathrm{uni}}$ the class of parameterized problems $(Q, \kappa)$ such that their complements $(\Sigma^* \setminus Q, \kappa)$ are in $\mathrm{XNL}_{\mathrm{uni}}$.

PROPOSITION 5.1.

*(a) If* $(Q,\kappa)$ *is slicewise downward monotone, then* $(Q,\kappa) \in \text{coXNL}_{\text{uni}}$.
*(b)* $\text{XNL}_{\text{uni}} \cap \text{coXNL}_{\text{uni}} \subseteq \text{XP}_{\text{uni}}$.

*Proof:* (a) Assume that $(Q,\kappa)$ is slicewise downward monotone and let $\mathbb{Q}$ be an algorithm enumerating the elements of $\Sigma^* \setminus Q$. Then the following algorithm shows that $(\Sigma^* \setminus Q,\kappa) \in \text{XNL}_{\text{uni}}$: If the input $y \in \Sigma^*$ has not the form $\langle x,n\rangle$ for some $x \in \Sigma^*$ and $n \in \mathbb{N}$, then it accepts. If $y = \langle x,n\rangle$, it guesses $\ell \in \mathbb{N}$; then it checks whether $\mathbb{Q}$ in its first $\ell$ steps enumerates some $\langle x,m\rangle$ with $m \leq n$; if so, it accepts.

(b) Let $(Q,\kappa) \in \text{XNL}_{\text{uni}} \cap \text{coXNL}_{\text{uni}}$. Then there is a nondeterministic Turing machine $\mathbb{M}$ and a function $f : \mathbb{N} \to \mathbb{N}$ such that for $x \in \Sigma^*$:

– if $x \in Q$, then no run of $\mathbb{M}$ on $x$ is rejecting and there is at least one accepting run using space $\leq f(\kappa(x)) \cdot \log |x|$;
– if $x \notin Q$, then no run of $\mathbb{M}$ on $x$ is accepting and there is at least one rejecting run using space $\leq f(\kappa(x)) \cdot \log |x|$.

We consider the following algorithm $\mathbb{A}$ deciding $Q$:

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

1. $\ell \leftarrow 1$
2. construct the graph of configurations of $\mathbb{M}$ on input $x$ using space $\leq \ell \cdot \log |x|$
3. **if** this graph contains an accepting run **then** accept
4. **if** this graph contains a rejecting run **then** reject
5. $\ell \leftarrow \ell + 1$
6. goto 2.

Clearly, $\mathbb{A}$ decides $Q$ and by our assumptions on $\mathbb{M}$ the algorithm $\mathbb{A}$ on input $x$ will halt for some $\ell \leq f(\kappa(x))$. For fixed $\ell$, Lines 2–4 take

$$2^{O(\ell \cdot \log |x|)} = |x|^{O(\ell)}$$

steps. Therefore the running time of $\mathbb{A}$ on input $x$ can be bounded by

$$\sum_{\ell=1}^{f(\kappa(x))} |x|^{O(\ell)} = |x|^{O(f(\kappa(x)))}. \qquad \square$$

Now we can show the following results extending Corollary 3.11 and complementing Corollary 4.8.

COROLLARY 5.2. *If* TAUT *has an almost space optimal nondeterministic algorithm, then it has an almost (time) optimal algorithm.*

*Proof:* By assumption and Theorem 3.10 (b) we have $p$-HALT$_>$ $\in$ $\text{XNL}_{\text{uni}}$ and thus $p$-HALT$_>$ $\in$ $\text{XNL}_{\text{uni}} \cap \text{coXNL}_{\text{uni}} \subseteq \text{XP}_{\text{uni}}$ by the previous proposition; now the claim follows from Theorem 3.3 (a). $\square$

COROLLARY 5.3. *If* $\text{TC}_{\text{inv}}$ *is an* NL*-bounded logic for* NL*, then* $\text{LFP}_{\text{inv}}$ *is a* P*-bounded logic for* P.

*Proof:* By assumption and Theorem 4.1 we have $p$-HALT$_>$ $\in$ $\text{XNL}_{\text{uni}}$ and thus $p$-HALT$_>$ $\in \text{XNL}_{\text{uni}} \cap \text{coXNL}_{\text{uni}} \subseteq \text{XP}_{\text{uni}}$; applying Theorem 4.1 again we get the claim. $\square$

To compare the complexity of parameterized problems here we use the notion of xlog-reduction: Let $(Q, \kappa)$ and $(Q', \kappa')$ be parameterized problems. We write $(Q, \kappa) \leq^{\text{xlog}} (Q', \kappa')$ if there is an *xlog-reduction* from $(Q, \kappa)$ to $(Q', \kappa')$, that is, a mapping $R : \Sigma^* \to \Sigma^*$ with:

(a) For all $x \in \Sigma^*$ we have $(x \in Q \iff R(x) \in Q')$.
(b) $R(x)$ is computable in space $f(\kappa(x)) \cdot \log |x|$ for some computable $f : \mathbb{N} \to \mathbb{N}$.
(c) There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

The parameterized problems $(Q, \kappa)$ and $(Q', \kappa')$ are *xlog-equivalent* if $(Q, \kappa) \leq^{\text{xlog}} (Q', \kappa')$ and $(Q', \kappa') \leq^{\text{xlog}} (Q, \kappa)$. These are notions of the usual (strongly uniform) parameterized complexity theory. For example, we get the corresponding notion $\leq^{\text{xlog}}_{\text{uni}}$ of uniform parameterized complexity by allowing the functions $f$ and $g$ (in (b) and (c), respectively) to be arbitrary and not necessarily computable.

As we haven't defined XL and XNL explicitly, we mention that we shall use part (b) of the following simple observation only for $\mathrm{C} = \mathrm{P}$.

LEMMA 5.4. *Let* $\mathrm{C} \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}\}$.

*(a) If* $(Q, \kappa) \leq^{\text{xlog}}_{\text{uni}} (Q', \kappa')$ *and* $(Q', \kappa') \in \mathrm{XC}_{\text{uni}}$*, then* $(Q, \kappa) \in \mathrm{XC}_{\text{uni}}$.
*(b) If* $(Q, \kappa) \leq^{\text{xlog}} (Q', \kappa')$ *and* $(Q', \kappa') \in \mathrm{XC}$*, then* $(Q, \kappa) \in \mathrm{XC}$.

We turn again to slicewise downward monotone problems. The goal of this section mentioned above can be stated in a precise form:

THEOREM 5.5. *Let* $\mathrm{C} \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$. *Then any two of the problems*

$$p\text{-FO-MODEL}_{>},\ p\text{-}L(\mathrm{C})\text{-MODEL}_{>},\ p\text{-FO-INV},\ p\text{-}L(\mathrm{C})\text{-INV},\ \textit{and}\ \ p\text{-HALT}_{>}$$

*are xlog-equivalent.*

Note that Lemma 4.7 is an immediate consequence of this theorem and Lemma 5.4 (a).

The rest of this section is devoted to a proof of Theorem 5.5. First we remark that among the slicewise downward monotone problems with underlying classical problem in coNP the problem $\text{HALT}_{>}$ is of highest complexity:

PROPOSITION 5.6. *Let* $(Q, \kappa)$ *be slicewise downward monotone and* $Q \in \mathrm{coNP}$. *Then*

$$(Q, \kappa) \leq^{\text{xlog}} p\text{-HALT}_{>}.$$

*Proof:* Let $\mathbb{M}$ be a nondeterministic Turing machine accepting the complement $\Sigma^* \setminus Q$ of $Q$. We may assume that for some $d \in \mathbb{N}$ the machine $\mathbb{M}$ on input $\langle x, n \rangle$ performs exactly $| \langle x, n \rangle |^d$ steps. For $x \in \Sigma^*$ let $\mathbb{M}_x$ be the nondeterministic Turing machine that on empty input tape, first writes $x$ on the tape, then guesses a natural number $m$, and finally it simulates the computation of $\mathbb{M}$ on input $\langle x, m \rangle$ and answers accordingly. We can assume that there is a logspace computable function $h$ such that $\mathbb{M}_x$ makes exactly $h(x, m) \in O\big(|x| + m + | \langle x, m \rangle |^d\big)$ steps if it guesses the natural number $m$. Furthermore we can assume that $h(x, m) < h(x, m')$ for $m < m'$.

Then $\langle x, n \rangle \mapsto \langle \mathbb{M}_x, h(x, n) \rangle$ is an xlog-reduction from $(Q, \kappa)$ to $p\text{-HALT}_{>}$: Clearly, if $\langle \mathbb{M}_x, h(x, n) \rangle \in p\text{-HALT}_{>}$, then by construction of $\mathbb{M}_x$ we have $\langle x, n \rangle \notin \Sigma^* \setminus Q$ and hence, $\langle x, n \rangle \in Q$. Conversely, if $\langle x, n \rangle \in Q$, then by slicewise downward monotonicity $\langle x, m \rangle \notin \Sigma^* \setminus Q$ for all $m \leq n$; thus $\langle \mathbb{M}_x, h(x, n) \rangle \in p\text{-HALT}_{>}$. □

Later on we shall use the following related result.

LEMMA 5.7. *Let $(Q,\kappa)$ be slicewise downward monotone and assume that there is a nondeterministic algorithm $\mathbb{A}$ accepting $\Sigma^* \setminus Q$ such that $t_{\mathbb{A}}(\langle x,n\rangle) \leq n^{f(|x|)}$ for some time constructible $f$ and all $\langle x,n\rangle \in \Sigma^* \setminus Q$. Then*

$$(Q,\kappa) \leq^{\text{xlog}} p\text{-HALT}_{>}.$$

*Proof:* Let $(Q',\kappa')$ be the problem

| | |
|---|---|
| *Instance:* | $x \in \Sigma^*$ and $1^m$ with $m \in \mathbb{N}$. |
| *Parameter:* | $\lvert x\rvert$. |
| *Question:* | Is $\langle x,n\rangle \in Q$ for all $n \in \mathbb{N}$ with $n^{f(\lvert x\rvert)} \leq m$? |

By the previous proposition we get our claim once we have shown:

(1) $(Q',\kappa')$ is slicewise downward monotone and $Q' \in$ coNP;
(2) $(Q,\kappa) \leq^{\text{xlog}} (Q',\kappa')$.

To see (1) let $\mathbb{A}$ be as stated above and let $\mathbb{T}$ be an algorithm witnessing the time constructibility of $f$; that is, $\mathbb{T}$ on input $1^k$ with $k \in \mathbb{N}$ computes $f(k)$ in exactly $f(k)$ steps. An algorithm $\mathbb{B}$ witnessing that $\Sigma^* \setminus Q' \in$ NP runs as follows on input $\langle x,m\rangle$:

– $\mathbb{B}$ guesses $n \in \mathbb{N}$;
– if $n = 1$, the algorithm $\mathbb{B}$ rejects in case $m = 0$;
if $n \geq 2$, the algorithm $\mathbb{B}$ simulates $m$ steps of the computation of $\mathbb{T}$ on input $1^{|x|}$; if thereby $\mathbb{T}$ does not stop, $\mathbb{B}$ rejects; otherwise, the simulation yields $f(|x|)$ and $\mathbb{B}$ checks whether $n^{f(|x|)} > m$ (this can be detected in time $O(m)$); in the positive case $\mathbb{B}$ rejects;
– finally $\mathbb{B}$ simulates the computation of $\mathbb{A}$ on $\langle x,n\rangle$ and answers accordingly.

(2) Note that the mapping $\langle x,n\rangle \mapsto \langle x, n^{f(|x|)}\rangle$ is an xlog-reduction. □

The following lemmas will finally yield a proof of Theorem 5.5.

LEMMA 5.8. *$p$-$L$-MODEL$_>$ $\leq^{\text{xlog}}$ $p$-$L$-INV for $L =$ FO or $L = L(\text{C})$ where C is one of the classes* L*,* NL*, and* P*.*

*Proof:* Let $\varphi$ be an $L[\tau]$-sentence. We set $\tau' := \tau \cup \{P\}$ with a new unary relation symbol $P$ and consider the $L[\tau'_<]$-sentence

$$\psi(\varphi) := \varphi \wedge \text{“}P \text{ holds for the first element of } <\text{.”}$$

If $\varphi$ has no model with exactly one element, then for $n \in \mathbb{N}$

$$\langle \varphi,n\rangle \in p\text{-}L\text{-MODEL}_{>} \iff \langle \psi(\varphi),n\rangle \in p\text{-}L\text{-INV}.$$

For such $\varphi$ we define the reduction by $\langle \varphi,n\rangle \mapsto \langle \psi(\varphi),n\rangle$ and by $\langle \varphi,n\rangle \mapsto \langle$“$P$ holds for the first element of $<$”, $2\rangle$ if the sentence $\varphi$ has a model with exactly one element. □

LEMMA 5.9. *$p$-LFP-INV $\leq^{\text{xlog}}$ $p$-HALT$_>$ (and hence $p$-$L$(C)-INV $\leq^{\text{xlog}}$ $p$-HALT$_>$ for* C = L *and* C = NL*).*

*Proof:* Consider the nondeterministic algorithm $\mathbb{A}$ that on input $\langle \varphi,m\rangle$, where $\varphi$ is an LFP-sentence and $m \geq 1$, guesses a structure $\mathcal{A}$ and two orderings $<_1$ and $<_2$, and accepts if $|A| \leq m$, $(\mathcal{A},<_1) \models_{\text{LFP}} \varphi$, and $(\mathcal{A},<_2) \models_{\text{LFP}} \neg\varphi$. It accepts the complement of

$p$-LFP-INV. As LFP is an effectively P-bounded logic for P on ordered structures, the algorithm $\mathbb{A}$ witnesses that $p$-LFP-INV satisfies the assumptions on $(Q, \kappa)$ in Lemma 5.7. This yields the claim. □

LEMMA 5.10. $p$-HALT$_>$ $\leq^{\text{xlog}}$ $p$-FO-MODEL$_>$.

*Proof:* Coding configurations of a nondeterministic machine and its nondeterministic choices in a standard way it is easy to assign to every nondeterministic Turing machine $\mathbb{M}$ a first-order sentence $\varphi_{\mathbb{M}}$ such that for all $n \in \mathbb{N}$

$$t_{\mathbb{M}}(\lambda) \leq n \iff \varphi_{\mathbb{M}} \text{ has a model with universe of cardinality } n.$$

Thus, $\langle \mathbb{M}, n\rangle \mapsto \langle \varphi_{\mathbb{M}}, n\rangle$ is the desired xlog-reduction. □

*Proof of Theorem 5.5:* Immediate by Lemmas 5.8, 5.9, and 5.10 □

## 6. OPTIMAL PROOF SYSTEMS, ALMOST OPTIMAL ALGORITHMS, AND LISTINGS

By [Krajíček and Pudlák 1989] the existence of an almost optimal algorithm for TAUT is equivalent to the existence of a polynomially optimal proof system for TAUT and by [Sadowski 2002] it is equivalent to the existence of listings of the subsets in P of TAUT. These equivalences extend to the different variants (strong, effective, and space) of almost optimal algorithms we considered in Section 3. In Section 6.1 we prove the strong variant; for the other cases we present the definitions and the results in Section 6.2.

*Listings.* Listings (or effective enumerations) of problems by means of Turing machines have been used to characterize promise classes possessing complete languages (e.g., see [Hartmanis and Hemachandra 1988; Kowalczyk 1984]). In the context of almost optimal algorithms, listings of subsets of TAUT have been used systematically by Sadowski (see [Sadowski 2002]). We introduce our general notion of listing.

DEFINITION 6.1. *Let* $Q \subseteq \Sigma^*$ *and* $\mathrm{C}, \mathrm{C}' \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}\}$

*(1) A* C-subset of $Q$ *is a set* $Y$ *with* $Y \subseteq Q$ *and* $Y \in \mathrm{C}$.

*(2) A* listing of the C-subsets of $Q$ by $\mathrm{C}'$-machines *is an algorithm* $\mathbb{L}$ *that, once having been started, eventually yields as outputs Turing machines* $\mathbb{M}_1, \mathbb{M}_2, \ldots$ *of type* $\mathrm{C}'$ *such that*

$$\big\{L(\mathbb{M}_i) \;\big|\; i \geq 1\big\} = \big\{Y \subseteq Q \;\big|\; Y \in \mathrm{C}\big\}.$$

*(3) Let* $\mathrm{C}, \mathrm{C}' \in \{\mathrm{P}, \mathrm{NP}\}$. *A listing* $\mathbb{L}$ *of the* C*-subsets of* $Q$ *by* $\mathrm{C}'$*-machines is* strong *if there is a constant* $d \in \mathbb{N}$ *such that for every* C*-subset* $Y$ *of* $Q$ *and every* $\mathrm{C}'$*-machine* $\mathbb{M}$ *deciding* $Y$ *there is a machine listed by* $\mathbb{L}$ *deciding* $x \in Y$ *in time* $\leq p(t_{\mathbb{M}}(x) + |x|)$, *where* $p \in \mathbb{N}_d[X]$ *(that is,* $p$ *is a polynomial of degree* $\leq d$*).*

*We write* LIST$(\mathrm{C}, Q, \mathrm{C}')$ *and* SLIST$(\mathrm{C}, Q, \mathrm{C}')$ *if there is a listing and a strong listing, respectively, of the* C*-subsets of* $Q$ *by* $\mathrm{C}'$*-machines.*

Sometimes we speak of the listing $\mathbb{M}_1, \mathbb{M}_2, \ldots$ (instead of the listing $\mathbb{L}$). By systematically adding polynomial time clocks (if $\mathrm{C}'$ is a time class) or devices controlling the space used, we may assume that *all* runs of the machines $\mathbb{M}_i$ on *any* input satisfy the time or space bound characteristic for $\mathrm{C}'$.

*Optimal proof systems.* Let $Q \subseteq \Sigma^*$. A *proof system for* $Q$ in the sense of [Cook and Reckhow 1979] is a polynomial time algorithm $\mathbb{P}$ computing a function from $\Sigma^*$ onto $Q$.

If $\mathbb{P}(w) = x$, we say that $w$ is a $\mathbb{P}$-*proof* of $x$. Often we introduce proof systems implicitly by defining the corresponding function; then this definition will suggest an algorithm.

Let $\mathbb{P}$ and $\mathbb{P}'$ be proof systems for $Q$. A *translation from* $\mathbb{P}'$ *into* $\mathbb{P}$ is a polynomial time algorithm $\mathbb{T}$ such that $\mathbb{P}(\mathbb{T}(w')) = \mathbb{P}'(w')$ for all $w' \in \Sigma^*$.

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 translation from $\mathbb{P}'$ into $\mathbb{P}$. If for $\mathbb{P}$ there is a constant $d \in \mathbb{N}$ such that for every proof system $\mathbb{P}'$ for $Q$ there is a translation $\mathbb{T}$ from $\mathbb{P}'$ into $\mathbb{P}$ and a $p \in \mathbb{N}_d[X]$ such that

$$t_{\mathbb{T}}(w') \le p\big(t_{\mathbb{P}'}(w') + |w'|\big)$$

for all $w' \in \Sigma^*$, then $\mathbb{P}$ is *strongly p-optimal*.

A proof system $\mathbb{P}$ for $Q$ is *optimal* if for every proof system $\mathbb{P}'$ for $Q$ and every $w' \in \Sigma^*$ there is a $w \in \Sigma^*$ such that $\mathbb{P}(w) = \mathbb{P}'(w')$ and $|w| \le |w'|^{O(1)}$. Again, if there is a constant $d \in \mathbb{N}$ such that for all $\mathbb{P}'$ and every $w' \in \Sigma^*$ there is a $w \in \Sigma^*$ such that $\mathbb{P}(w) = \mathbb{P}'(w')$ and $|w| \le p\big(t_{\mathbb{P}'}(w') + |w'|\big)$ for some $p \in \mathbb{N}_d[X]$, then $\mathbb{P}$ is *strongly optimal*.

In the following theorem we summarize the results that are known for $Q = \text{TAUT}$ mentioned at the beginning of this section. The extensions to arbitrary $Q$ with padding are mainly due to Messner [Messner 2000] or are implicit in [Sadowski 2002].

THEOREM 6.2.

*(1) For every $Q$ we have (a) $\Rightarrow$ (b) and (b) $\Rightarrow$ (c); moreover (a), (b), and (c) are all equivalent if $Q$ has padding. Here*
- *(a) $Q$ has a p-optimal proof system.*
- *(b) $Q$ has an almost optimal algorithm.*
- *(c) $\text{LIST}(\text{P}, Q, \text{P})$.*

*(2) For every $Q$ we have (a) $\iff$ (b) and (b) $\Rightarrow$ (c); moreover (a)–(c) are all equivalent if $Q$ has padding. Here*
- *(a) $Q$ has an optimal proof system.*
- *(b) $Q$ has an almost optimal nondeterministic algorithm.*
- *(c) $\text{LIST}(\text{NP}, Q, \text{NP})$.*

### 6.1. The strong variant

The following series of lemmas will lead to a proof of the result corresponding to Theorem 6.2 for the strong notions.

LEMMA 6.3. *Let $Q \subseteq \Sigma^*$. If there is a strongly almost optimal (nondeterministic) algorithm for $Q$, then* $\text{SLIST}(\text{P}, Q, \text{P})$ $\big(\text{SLIST}(\text{NP}, Q, \text{NP})\big)$.

*Proof:* We prove the deterministic case, the nondeterministic one is obtained by obvious changes. Let $\mathbb{O}$ be a strongly almost optimal algorithm for $Q$. Let $d \in \mathbb{N}$ bound the degree of the polynomials $p$ in the definition of strongly almost optimal (Definition 3.1). Let $t : \Sigma^* \to \{1\}^*$ be a function such that $t = t_{\mathbb{M}}$ for some Turing machine $\mathbb{M}$ (here we identify $1^k$ with $k$). Let $\mathbb{O}(t)$ be the algorithm that on input $x$ simulates $\mathbb{O}$ on input $x$ but rejects if the simulation exceeds time $t(x)$. Clearly,

(i) $L(\mathbb{O}(t))$ is a P-subset of $Q$ if $t$ is computable in polynomial time.

Moreover:

(ii) For every P-subset $Y$ of $Q$ and every polynomial time Turing machine $\mathbb{M}$ deciding $Y$ there is a $p \in \mathbb{N}_d[X]$ with $Y \subseteq L(\mathbb{O}(p[t_{\mathbb{M}}]))$, where $p[t_{\mathbb{M}}] : \Sigma^* \to \{1\}^*$ is defined by

$$p[t_{\mathbb{M}}](x) := p\big(t_{\mathbb{M}}(x) + |x|\big).$$

In fact, we consider the following algorithm $\mathbb{C}$ which decides $Q$: On input $x$, in parallel it simulates $\mathbb{M}$ and $\mathbb{O}$ on input $x$; if $\mathbb{M}$ halts first and accepts, then $\mathbb{C}$ accepts, otherwise it answers as $\mathbb{O}$. Note that $t_{\mathbb{C}}(x) \leq O(t_{\mathbb{M}}(x))$ for $x \in Y$. By the strongly almost optimality of $\mathbb{O}$ there is a $p \in \mathbb{N}_d[X]$ with

$$t_{\mathbb{O}}(x) \leq p\big(t_{\mathbb{C}}(x) + |x|\big)$$

for all $x \in Q$ and thus for all $x \in Y$,

$$t_{\mathbb{O}}(x) \leq p\big(O(t_{\mathbb{M}}(x)) + |x|\big).$$

Therefore, $Y \subseteq L(\mathbb{O}(p_1[t_{\mathbb{M}}]))$ for some $p_1 \in \mathbb{N}_d[X]$.

We fix a listing $\mathbb{M}_1, \mathbb{M}_2, \ldots$ of all polynomial time deterministic Turing machines. By (i)–(ii), $\big(\mathbb{M}_i(\mathbb{O}(p[t_{\mathbb{M}_i}]))\big)_{i\geq 1,\ p\in\mathbb{N}_d[X]}$ is a strong listing of the P-subsets of $Q$, where $\mathbb{M}_i(\mathbb{O}(p[t_{\mathbb{M}_i}]))$ on input $x$, first simulates $\mathbb{O}(p[t_{\mathbb{M}_i}])$ on $x$ and if this algorithm accepts, then it simulates $\mathbb{M}_i$ on input $x$ and answers accordingly. □

In a second step we deal with the transition from listings to optimal proof systems.

LEMMA 6.4. *Assume that $Q$ has a padding function.*

*(a) If* $\mathrm{SLIST}(\mathrm{P}, Q, \mathrm{P})$*, then $Q$ has a strongly p-optimal proof system.*
*(b) If* $\mathrm{SLIST}(\mathrm{NP}, Q, \mathrm{NP})$*, then $Q$ has a strongly optimal proof system.*

*Proof:* Again we prove (a), thereby indicating the changes that are necessary for a proof of (b). Fix $y_0 \in Q$ and let $\mathit{pad} : \Sigma^* \times \Sigma^* \to \Sigma^*$ be a padding function for $Q$. Let $\mathbb{L}$ be a strong listing of the P-subsets of $Q$ by P-machines and let $d \in \mathbb{N}$ be such that polynomials in $\mathbb{N}_d[X]$ witness the strongness of $\mathbb{L}$. We say that $v \in \Sigma^*$ is a *proof string* if it has the form

$$v = \big\langle \mathbb{M}, w, y, 1^m, \mathbb{M}', 1^\ell, 1^r, 1^t \big\rangle,$$

where

– $\mathbb{M}$ is a Turing machine that on input $w$ outputs $y$ in $\leq m$ steps;
– $\mathbb{L}$ lists $\mathbb{M}'$ in $\leq \ell$ steps;
– $\mathbb{M}'$ accepts $\mathit{pad}(y, \langle w, 1^r\rangle)$ in $\leq t$ steps. (For (b) we have to add $s$ in the tuple $v$, where $s$ is the sequence of states of a run of $\mathbb{M}'$ accepting $\mathit{pad}(y, \langle w, 1^r\rangle)$ in $\leq t$ steps.)

Clearly, we can decide in polynomial time whether a string $v \in \Sigma^*$ is a proof string. Moreover, if $v$ is a proof string, then $y \in Q$ (as $L(\mathbb{M}') \subseteq Q$ and $\mathbb{M}'$ accepts $\mathit{pad}(y, \langle w, 1^r\rangle)$). We consider the proof system $\mathbb{P}$ defined by

$$\mathbb{P}\big(\big\langle \mathbb{M}, w, y, 1^m, \mathbb{M}', 1^\ell, 1^r, 1^t \big\rangle\big) := y$$

if $\big\langle \mathbb{M}, w, y, 1^m, \mathbb{M}', 1^\ell, 1^r, 1^t \big\rangle$ is a proof string, and $\mathbb{P}(w) := y_0$ otherwise. Clearly, $\mathbb{P}$ is polynomial time computable and has a subset of $Q$ as range. So, $\mathbb{P}$ is a proof system for $Q$ if we can show that every $y \in Q$ is in its range: As $\{\mathit{pad}(y, \langle y, 1\rangle)\}$ is a P-subset of $Q$, a machine $\mathbb{M}'$ deciding $\{\mathit{pad}(y, \langle y, 1\rangle)\}$ is listed by $\mathbb{L}$, say, in $\ell$ steps. Then $\mathbb{P}(v) = y$ for $v = \big\langle \mathbb{M}_{\mathrm{id}}, y, y, 1^m, \mathbb{M}', 1^\ell, 1, 1^t \big\rangle$, where $\mathbb{M}_{\mathrm{id}}$ is a machine that on input $x$ outputs $x$ and that on input $y$ takes $m$ steps, and $t$ is the number of steps of $\mathbb{M}'$ on input $\mathit{pad}(y, \langle y, 1\rangle)$.

It remains to show that $\mathbb{P}$ is strongly p-optimal. For this purpose let $\mathbb{P}' : \Sigma^* \to Q$ be a proof system for $Q$. Then,

$$\mathit{Graph}(\mathbb{P}') := \big\{\mathit{pad}(y, \langle w', 1^r\rangle) \;\big|\; y, w' \in \Sigma^*, \mathbb{P}'(w') = y \text{ and } r = t_{\mathbb{P}'}(w')\big\}$$

is a P-subset of $Q$. Let $\mathbb{B}$ be the algorithm deciding $\mathit{Graph}(\mathbb{P}')$, which on input $x$ first applying property (ii) of the padding function (see page 5) computes $w'$ and $r$ if $x = \mathit{pad}(\ldots, \langle w', 1^r\rangle)$, then it computes $y := \mathbb{P}'(w')$ and checks if $r = t_{\mathbb{P}'}(w')$, both by simulating $\mathbb{P}'$ (for at most $r$ steps), and finally it checks whether $x = \mathit{pad}(y, \langle w', 1^r\rangle)$. As $r, |y| \leq O(|x|)$, there is a polynomial $q_0$ depending on the functions $\mathit{pad}$ and $\langle\ ,\ \rangle$ only, say of degree $d_0 \geq 1$, such that

$$t_{\mathbb{B}}(x) \leq q_0(|x|).$$

Hence, by assumption, $\mathbb{L}$ lists a machine $\mathbb{M}'$, say in $\ell$ steps, that decides $x \in \mathit{Graph}(\mathbb{P}')$ in $\leq p\big(q_0(|x|)\big)$ steps for some $p \in \mathbb{N}_d[X]$ $\big($as $t_{\mathbb{B}}(x) + |x| \leq 2 \cdot q_0(|x|)\big)$. Furthermore note that for $w' \in \Sigma^*$ and $x := \mathit{pad}(\mathbb{P}'(w'), \langle w', 1^{t_{\mathbb{P}'}(w')}\rangle)$ we have

$$|x| \leq q_1(t_{\mathbb{P}'}(w') + |w'|),$$

where again the polynomial $q_1$, say of degree $d_1$, depends only on the padding function and the tupling function. We define the translation $\mathbb{T}$ by

$$\mathbb{T}(w') := \Big\langle \mathbb{P}', w', \mathbb{P}'(w'), 1^{t_{\mathbb{P}'}(w')}, \mathbb{M}', 1^{\ell}, 1^{t_{\mathbb{P}'}(w')}, 1^{t} \Big\rangle \tag{15}$$

where

$$t := p\Big(q_0\big(q_1(t_{\mathbb{P}'}(w') + |w'|)\big)\Big).$$

It is easy to check that $\mathbb{T}(w')$ is a proof string with $\mathbb{P}(\mathbb{T}(w')) = \mathbb{P}'(w')$. As $\mathbb{M}'$ and $1^{\ell}$ do not depend on $w'$, the algorithm $\mathbb{T}$ on $w'$ needs $p_1\big(t_{\mathbb{P}'}(w') + |w'|\big)$ steps for a polynomial $p_1$ of degree $\leq d + d_0 + d_1$; this finishes the proof. For (b) we get a tuple $w$ with the desired properties by adding to the tuple in (15) a sequence $s$ of states of a run of $\mathbb{M}'$ accepting $\mathit{pad}\big(\mathbb{P}'(w'), w'\big)$ of length $\leq t$. □

The reader will have noticed that the previous proof also shows the conclusion of part (b) under the hypothesis $\textsc{SList}(\mathrm{P}, Q, \mathrm{NP})$ as the set $\mathit{Graph}(\mathbb{P}')$ is in P. However, the equivalence $\big(\textsc{SList}(\mathrm{P}, Q, \mathrm{NP}) \iff \textsc{SList}(\mathrm{NP}, Q, \mathrm{NP})\big)$ holds for all $Q$ with padding. This can be shown along the lines of the proof of Proposition 6.12, where the "non-strong" version of this equivalence is derived.

We turn to the last step, the transition from proof systems to almost optimal algorithms. We need the following result on inverters due to Levin [Levin 1973].

DEFINITION 6.5. *Let $f : \Sigma^* \to \Sigma^*$ be a function. An algorithm $\mathbb{A}$* inverts *$f$ if for every $x$ in the* range *of $f$ the algorithm $\mathbb{A}$ computes some $w$ with $f(w) = x$. For $x$ not in the range of $f$ the algorithm $\mathbb{A}$ does not halt.*

By Levin's result, for any algorithm $\mathbb{F}$ computing a function $f$ there is an inverter $\mathbb{O}$, which is optimal with respect to the time required by the computation of $\mathbb{F}(\mathbb{O}(y))$.

THEOREM 6.6. *There is a $d_0 \in \mathbb{N}$ such that for all algorithms $\mathbb{F}$ computing a function $f : \Sigma^* \to \Sigma^*$ there exists an algorithm $\mathbb{O}$ such that:*

*(a) $\mathbb{O}$ inverts $f$;*
*(b) for every algorithm $\mathbb{I}$ inverting $f$ there is an $p \in \mathbb{N}_{d_0}[X]$ such that for every $y$ in the range of $f$ we have*

$$t_{\mathbb{O}}(y) \leq p\big(|y| + t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))\big).$$

We now state the result yielding an almost optimal algorithm from a p-optimal proof system.

LEMMA 6.7.

*(a) If $Q$ has a strongly p-optimal proof system, then $Q$ has a strongly almost optimal algorithm.*
*(b) If $Q$ has a strongly optimal proof system, then $Q$ has a strongly almost optimal nondeterministic algorithm.*

*Proof:* Again we only prove (a), then (b) is obtained by changes similar to that indicated in the previous proof for the nondeterministic case. Let $\mathbb{P} : \Sigma^* \to Q$ be a strongly p-optimal proof system for $Q$ and let $d \in \mathbb{N}$ be a bound for the degrees of the polynomials according to the strongness. Fix $y_0 \in Q$. We define the function $f : \Sigma^* \to \Sigma^*$ by

$$f\big(\langle \mathbb{A}, y, 1^m, \mathbb{S}, 1^t \rangle\big) := y$$

if

(1) $\mathbb{A}$ is an algorithm that accepts $y \in \Sigma^*$ in $\leq m$ steps;
(2) $\mathbb{S}$ is an algorithm that on input $\langle y, 1^m \rangle$ computes a string $w$ with $\mathbb{P}(w) = y$ in $\leq t$ steps.

Otherwise, we set $f(w) := y_0$. It is easy to verify that the range of $f$ is $Q$, in particular (2) guarantees that it is a subset of $Q$. Moreover there is an algorithm $\mathbb{F}$ that computes the function $f$ in polynomial time, say, in time $\leq q_{\mathbb{F}}(n)$ with $q_{\mathbb{F}}$ of degree $d(\mathbb{F})$. We choose $d_0$ according to Theorem 6.6 and an optimal inverter $\mathbb{O}$, that is, an inverter with the properties (a) and (b) of Theorem 6.6; in particular, for every inverter $\mathbb{I}$ of $f$ there is a $p \in \mathbb{N}_{d_0}[X]$ such that for every $y \in Q$

$$t_{\mathbb{O}}(y) \leq p\big(|y| + t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))\big) \leq p_1(|y| + t_{\mathbb{I}}(y)) \tag{16}$$

with $p_1 \in \mathbb{N}_{d_0+d(\mathbb{F})}[X]$ as $t_{\mathbb{F}}(\mathbb{I}(y)) \leq q_{\mathbb{F}}(|\mathbb{I}(y)|) \leq q_{\mathbb{F}}(t_{\mathbb{I}}(y))$.

Let $\mathbb{Q}$ be any algorithm deciding $Q$. We claim that the following algorithm $\mathbb{O}\|\mathbb{Q}$ is a strongly almost optimal algorithm deciding $Q$, where $\mathbb{O}\|\mathbb{Q}$ on input $y$ simulates $\mathbb{O}$ and $\mathbb{Q}$ on input $y$ in parallel; if $\mathbb{O}$ halts first, then it accepts, and if $\mathbb{Q}$ halts first, then it answers accordingly.

The algorithm $\mathbb{O}\|\mathbb{Q}$ decides $Q$ (here we use Theorem 6.6 (a)) and for $y \in Q$ we have

$$t_{\mathbb{O}\|\mathbb{Q}}(y) \leq O(t_{\mathbb{O}}(y)). \tag{17}$$

We claim that $\mathbb{O}\|\mathbb{Q}$ is strongly almost optimal. For this purpose let $\mathbb{A}$ be any algorithm that decides $Q$. We get a proof system $\mathbb{P}'_{\mathbb{A}}$ for $Q$ by setting

$$\mathbb{P}'_{\mathbb{A}}(w') := \begin{cases} y, & \text{if } w' = \langle y, 1^m \rangle \text{ and } \mathbb{A} \text{ accepts } y \text{ in} \leq m \text{ steps} \\ y_0, & \text{otherwise.} \end{cases}$$

Then, $t_{\mathbb{P}'_{\mathbb{A}}}(w') = O(|w'|)$. Since $\mathbb{P}$ is strongly p-optimal for $Q$, there is a translation $\mathbb{T}$ and a polynomial $p_2 \in \mathbb{N}_d[X]$ such that for all $w' \in \Sigma^*$

$$\mathbb{P}(\mathbb{T}(w')) = \mathbb{P}'_{\mathbb{A}}(w') \text{ and } t_{\mathbb{T}}(w') \leq p_2\big(t_{\mathbb{P}'_{\mathbb{A}}}(w') + |w'|\big) \leq p_2(O(|w'|)). \tag{18}$$

Using $\mathbb{A}$ and $\mathbb{T}$ we define an inverter $\mathbb{I}$ of the function $f$: On input $y$, the algorithm $\mathbb{I}$ simulates the algorithm $\mathbb{A}$ on $y$; if $\mathbb{A}$ rejects $y$, then $\mathbb{I}$ does not halt, if $\mathbb{A}$ accepts $y$, then it outputs

$$\Big\langle \mathbb{A}, y, 1^{t_{\mathbb{A}}(y)}, \mathbb{T}, 1^t \Big\rangle \text{ with } t := t_{\mathbb{T}}\big(\langle y, 1^{t_{\mathbb{A}}(y)} \rangle\big).$$

Thus, for $y \in Q$

$$\begin{aligned} t_{\mathbb{I}}(y) &\leq O\Big(t_{\mathbb{A}}(y) + t_{\mathbb{T}}\big(\langle y, 1^{t_{\mathbb{A}}(y)} \rangle\big)\Big) \\ &= O\Big(t_{\mathbb{A}}(y) + p_2\big(O(|y| + t_{\mathbb{A}}(y))\big)\Big) \qquad \text{(by (18))}. \end{aligned}$$

Hence by (16),

$$t_{\mathbb{O}}(y) \le p_1(|y| + t_{\mathbb{I}}(y)) \le p_1\Big(|y| + O\Big(t_{\mathbb{A}}(y) + p_2(O(|y| + t_{\mathbb{A}}(y)))\Big)\Big)$$
$$\le p_3(|y| + t_{\mathbb{A}}(y))$$

for some $p_3 \in \mathbb{N}_{d_0+d(\mathbb{F})+d}[X]$. Together with (17), this shows the strongly almost optimality of $\mathbb{O}\|\mathbb{Q}$. □

It is well-known that nondeterministic algorithms and proof systems are more or less the same. We use this fact to show that for arbitrary $Q$ a strongly almost optimal nondeterministic algorithm yields a strongly optimal proof system. By the way, we could have already used this fact to get a simpler proof of part (b) of the previous lemma, a proof tailored only for the nondeterministic case.

LEMMA 6.8. *If $Q$ has a strongly almost optimal nondeterministic algorithm, then it has a strongly optimal proof system.*

*Proof:* Let $\mathbb{A}$ be a strongly almost optimal nondeterministic algorithm for $Q$ and let $d \ge 1$ be such that polynomials in $\mathbb{N}_d[X]$ witness the strongness of $\mathbb{A}$. We fix $y_0 \in Q$ and define the proof system $\mathbb{P}$ for $Q$ by

$$\mathbb{P}(w) := \begin{cases} y, & \text{if } w = \langle y, s\rangle, \text{ where } s \text{ is the sequence of states of a run of } \mathbb{A} \text{ accepting } y \\ y_0, & \text{otherwise.} \end{cases}$$

The proof system $\mathbb{P}$ is strongly optimal: Let $\mathbb{P}'$ be any proof system for $Q$. The nondeterministic algorithm $\mathbb{B}(\mathbb{P}')$ accepts $Q$, where $\mathbb{B}(\mathbb{P}')$ on input $y \in \Sigma^*$ guesses a string $w'$ and accepts if $\mathbb{P}'(w') = y$. Hence, there is a $q \in \mathbb{N}_d[X]$ such that for all $y \in Q$

$$t_{\mathbb{A}}(y) \le q\big(t_{\mathbb{B}(\mathbb{P}')}(y) + |y|\big). \tag{19}$$

Let $\mathbb{P}'(w') = y$. Then

$$t_{\mathbb{B}(\mathbb{P}')}(y) = O\big(|w'| + t_{\mathbb{P}'}(w')\big). \tag{20}$$

Let $s$ be the sequence of states of a run of $\mathbb{A}$ accepting $y$ in $t_{\mathbb{A}}(y)$ steps and set $w := \langle y, s\rangle$. Then $\mathbb{P}(w) = y$ and

$$|w| = \big|\langle y, s\rangle\big| = O\big(|y| + |s|\big) = O(|y| + t_{\mathbb{A}}(y)) = O\Big(t_{\mathbb{P}'}(w') + q\big(t_{\mathbb{B}(\mathbb{P}')}(y) + |y|\big)\Big) \quad \text{(by (19))}$$
$$= O\Big(t_{\mathbb{P}'}(w') + q\big(O\big(|w'| + t_{\mathbb{P}'}(w')\big) + t_{\mathbb{P}'}(w')\big)\Big) \quad \text{(by (20)).}$$

This shows that $|w| \le p_1(t_{\mathbb{P}'}(w') + |w'|)$ for some $p_1 \in \mathbb{N}_d[X]$. □

Summarizing we have shown:

THEOREM 6.9.

*(1) For every $Q$ we have (a) $\Rightarrow$ (b) and (b) $\Rightarrow$ (c); moreover (a), (b), and (c) are all equivalent if $Q$ has padding. Here*
- *(a) $Q$ has a strongly p-optimal proof system.*
- *(b) $Q$ has an strongly almost optimal algorithm.*
- *(c) $\mathrm{SLIST}(\mathrm{P}, Q, \mathrm{P})$.*

*(2) For every $Q$ we have (a) $\iff$ (b) and (b) $\Rightarrow$ (c); moreover (a)–(c) are all equivalent if $Q$ has padding. Here*
- *(a) $Q$ has a strongly optimal proof system.*
- *(b) $Q$ has a strongly almost optimal nondeterministic algorithm.*
- *(c) $\mathrm{SLIST}(\mathrm{NP}, Q, \mathrm{NP})$.*

### 6.2. Further variants

Here we present the space variant and the effective variant of the results of Theorem 6.2.

*The space variant*. First we introduce the notion of space optimal logspace proof system (for simplicity, only the "deterministic optimality") and state the result afterwards.

DEFINITION 6.10.

*(1) A* logspace proof system *for $Q$ is a logspace algorithm $\mathbb{P}$ computing a function from $\Sigma^*$ onto $Q$.*
*(2) A logspace proof system $\mathbb{P}$ for $Q$ is* space optimal *if for every logspace proof system $\mathbb{P}'$ there is a translation from $\mathbb{P}'$ into $\mathbb{P}$ computable in logspace.*

In [Messner and Torán 1998] the authors introduced the notions of logspace proof system and logspace translation. They showed the equivalence of the statements "p-optimal proof systems exist," "p-optimal proof systems with respect to logspace translation exist," and "optimal logspace proof systems with respect to polynomial time translation exist." Using this fact, they showed that various complexity classes contain problems complete under logspace reductions if p-optimal proof systems exist.

THEOREM 6.11. *Assume $Q \subseteq \Sigma^*$ has padding. The following are equivalent:*

*(1) The following are equivalent:*
  *(a) $Q$ has a space optimal logspace proof system.*
  *(b) $Q$ has an almost space optimal algorithm.*
  *(c)* $\textsc{List}(\mathrm{L}, Q, \mathrm{L})$.
*(2) The following are equivalent:*
  *(a) $Q$ has an almost space optimal nondeterministic algorithm.*
  *(b)* $\textsc{List}(\mathrm{NL}, Q, \mathrm{NL})$.

*Proof:* Part (1) and the implication from (a) to (b) in part (2) can be obtained along the lines of our proof of Theorem 6.9 in Section 6.1; however, for the implication from (a) to (b) of part (1), we need a space version of Levin's result, which we state and prove in Section 6.3 as we haven't found it in the literature.

(2) (b) $\Rightarrow$ (a): Let $\mathbb{L}$ a listing witnessing that $\textsc{List}(\mathrm{NL}, Q, \mathrm{NL})$. It should be clear that the following nondeterministic algorithm $\mathbb{O}$ accepts $Q$.

> $\mathbb{O}$ // $x \in \Sigma^*$
>
> 1. guess an $i \in \mathbb{N}$ and compute the $i$th machine $\mathbb{M}_i$ listed by $\mathbb{L}$
> 2. guess a $d \in \mathbb{N}$ (in binary)
> 3. simulate $\mathbb{M}_i$ on $\mathit{pad}(x, x01^d)$ and output accordingly.

Whenever during the simulation of $\mathbb{M}_i$ on $\mathit{pad}(x, x01^d)$ a bit of the input $\mathit{pad}(x, x01^d)$ is required, the algorithm $\mathbb{O}$ simulates the computation of $\mathit{pad}(x, x01^d)$ till this bit is obtained. We show that $\mathbb{O}$ is almost space optimal. To that end, let $\mathbb{A}$ be a nondeterministic algorithm accepting $Q$. We consider the subset $\mathit{LOG}(\mathbb{A})$ of $Q$, where

$$\mathit{LOG}(\mathbb{A}) := \{\mathit{pad}(x, x01^d) \mid \ d \in \mathbb{N} \text{ and } \mathbb{A} \text{ accepts } x \text{ using space at most } \log d\}.$$

By the properties of a padding function it is easy to show that $\mathit{LOG}(\mathbb{A}) \in \mathrm{NL}$. Therefore, there exists an $i_0 \in \mathbb{N}$ such that the $i_0$th machine $\mathbb{M}_{i_0}$ listed by $\mathbb{L}$ accepts $\mathit{LOG}(\mathbb{A})$ in

space $O(\log n)$; in particular,

$$s_{\mathbb{M}_{i_0}}\big(\mathit{pad}(x, x01^d)\big) = O\big(\log |\mathit{pad}(x, x01^d)|\big) \begin{array}{l} = O\left(\log\left(|x|^{O(1)} + d^{O(1)}\right)\right) \\ = O(\log |x| + \log d). \end{array} \tag{21}$$

The second equality holds as *pad* is computable in logspace and hence, in polynomial time. Let $x \in Q$. We consider the run of the algorithm $\mathbb{O}$ on input $x$, where it guesses $i := i_0$ (in Line 1) and $d := 2^{s_{\mathbb{A}}(x)}$(in Line 2). These choices show that

$$s_{\mathbb{O}}(x) \leq O\big(c + s_{\mathbb{A}}(x) + \log |x| + \log \big|\mathit{pad}(x, x01^d)\big| + s_{\mathbb{M}_{i_0}}(\mathit{pad}(x, x01^d))\big), \tag{22}$$

where $c$ counts the space for guessing $i_0$ and for computing the machine $\mathbb{M}_{i_0}$. By (21) and (22) we conclude that

$$s_{\mathbb{O}}(x) \leq O(s_{\mathbb{A}}(x) + \log |x|). \qquad \square$$

We will use the following simple observations on listings to generalize some results of the preceding sections from $Q = \text{TAUT}$ to arbitrary $Q$ with padding. For $Q = \text{TAUT}$, part (c) of the next proposition has already been shown in [Sadowski 2002] by completely different means. Recall that $\mathrm{C}, \mathrm{C}', \ldots$ range over the complexity classes L, NL, P and NP.

PROPOSITION 6.12.

*(a) Let* C, C′, *and* C″ *be complexity classes with* $\mathrm{C}' \subseteq \mathrm{C}''$. *If* $\text{LIST}(\mathrm{C}, Q, \mathrm{C}')$, *then* $\text{LIST}(\mathrm{C}, Q, \mathrm{C}'')$.

*(b) Let* C, C′, *and* $\mathrm{C}_0$ *be complexity classes with* $\mathrm{C}_0 \subseteq \mathrm{C} \subseteq \mathrm{C}'$. *If* $\text{LIST}(\mathrm{C}, Q, \mathrm{C}')$, *then* $\text{LIST}(\mathrm{C}_0, Q, \mathrm{C}')$.

*(c) Assume* $Q$ *has padding. Then*

$$\text{LIST}(\mathrm{NP}, Q, \mathrm{NP}) \iff \text{LIST}(\mathrm{P}, Q, \mathrm{NP}).$$

*In particular,* $\text{LIST}(\mathrm{P}, Q, \mathrm{P})$ *implies* $\text{LIST}(\mathrm{NP}, Q, \mathrm{NP})$.

*Proof:* (a) is trivial. (b) Let $\mathbb{M}'$ be a Turing machine of type C′ and $\mathbb{M}_0$ one of type $\mathrm{C}_0$. Let $\mathbb{M}'(\mathbb{M}_0)$ be the Turing machine that on input $x$, first, by brute force, checks whether $\mathbb{M}'$ and $\mathbb{M}_0$ accept the same strings of length $\leq \log\log|x|$; if so, then it simulates $\mathbb{M}'$ on $x$ (and answers accordingly), otherwise it rejects. One easily verifies that $\mathbb{M}'(\mathbb{M}_0)$ is a machine of type C′; furthermore

$$L\big(\mathbb{M}'(\mathbb{M}_0)\big) = \begin{cases} L(\mathbb{M}'), & \text{if } L(\mathbb{M}_0) = L(\mathbb{M}') \\ \text{a finite subset of } L(\mathbb{M}'), & \text{otherwise.} \end{cases}$$

Therefore, if $\mathbb{M}'_1, \mathbb{M}'_2, \ldots$ is a listing of the C-subsets of $Q$ by C′-machines and $\mathbb{M}_1, \mathbb{M}_2, \ldots$ an enumeration of all Turing machines of type $\mathrm{C}_0$, then the listing $\big(\mathbb{M}'_i(\mathbb{M}_j)\big)_{i\geq 1,\ j\geq 1}$ witnesses that $\text{LIST}(\mathrm{C}_0, Q, \mathrm{C}')$.

(c) Let *pad* be a padding function for $Q$. By (b) it suffices to show the implication from right to left. Hence, we assume that $\text{LIST}(\mathrm{P}, Q, \mathrm{NP})$. For a nondeterministic Turing machine $\mathbb{M}$, we set

$$\mathit{Comp}(\mathbb{M}) := \big\{\mathit{pad}(x, \langle x, c\rangle) \;\big|\; x \in \Sigma^* \text{ and } c \text{ is a computation } [3] \text{ of } \mathbb{M} \text{ accepting } x\big\}.$$

Clearly, $\mathit{Comp}(\mathbb{M}) \in \mathrm{P}$; moreover

$$\mathit{Comp}(\mathbb{M}) \subseteq Q \iff L(\mathbb{M}) \subseteq Q. \tag{23}$$

[3] That is $c$ is the sequence of configurations of a run of $\mathbb{M}$ on $x$.

Hence, if $\mathbb{M}_1, \mathbb{M}_2, \ldots$ is a listing of the P-subsets of $Q$ by NP-machines, then $\mathbb{M}_1^*, \mathbb{M}_2^*, \ldots$ is a listing of the NP-subsets of $Q$ by NP-machines, where $\mathbb{M}_i^*$ on input $x$ guesses a string $c$ and simulates $\mathbb{M}_i$ on input $\mathit{pad}(x, \langle x, c\rangle)$. □

Now we can prove Corollary 3.7 (a) for arbitrary $Q$ with padding:

COROLLARY 6.13. *Assume that $Q$ has padding. If $Q$ has an almost optimal algorithm, then it also has an almost optimal nondeterministic algorithm.*

*Proof:* If $Q$ has an almost optimal algorithm, then $\mathrm{LIST}(\mathrm{P}, Q, \mathrm{P})$ by Theorem 6.2. Therefore $\mathrm{LIST}(\mathrm{NP}, Q, \mathrm{NP})$ by part (c) of the previous proposition. Applying again Theorem 6.2 we get the claim. □

We were unable to prove the logspace analogue of Proposition 6.12 (c), however, as a byproduct of the main result of Section 7 relating listings and logics (Theorem 7.1), we will get the logspace analogue of (c) for $Q = \mathrm{TAUT}$, that is, $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{NL})$ implies $\mathrm{LIST}(\mathrm{NL}, \mathrm{TAUT}, \mathrm{NL})$.

For the set $\mathit{Comp}(\mathbb{M})$ introduced for any nondeterministic Turing machine in the proof of (c) of the previous proposition, we even have $\mathit{Comp}(\mathbb{M}) \in \mathrm{L}$. We use this to obtain the following observation.

PROPOSITION 6.14.

*(a) If $Q$ has padding and $\mathrm{LIST}(\mathrm{L}, Q, \mathrm{P})$, then $\mathrm{LIST}(\mathrm{P}, Q, \mathrm{P})$.*
*(b) If $Q$ has padding and $\mathrm{LIST}(\mathrm{NL}, Q, \mathrm{NL})$, then $\mathrm{LIST}(\mathrm{NP}, Q, \mathrm{NP})$.*

*Proof:* (a) For deterministic Turing machines $\mathbb{M}$ and $\mathbb{M}'$ let $\mathbb{M}(\mathbb{M}')$ be the Turing machine that on input $x$, first by simulating $\mathbb{M}'$ on input $x$ stores its sequence $c$ of configurations and then runs $\mathbb{M}$ on input $\mathit{pad}(x, \langle x, c\rangle)$. By (23), if $\mathbb{M}_1, \mathbb{M}_2, \ldots$ is a listing of the L-subsets of $Q$ by P-machines and $\mathbb{M}'_1, \mathbb{M}'_2, \ldots$ an enumeration of all polynomial time Turing machines, then the enumeration $\big(\mathbb{M}_i(\mathbb{M}'_j)\big)_{i\geq 1,\ j\geq 1}$ witnesses that $\mathrm{LIST}(\mathrm{P}, Q, \mathrm{P})$.
The proof of (b) is obtained by obvious modifications. □

As a corollary of this proposition we get, using Theorem 6.2 and Theorem 6.9, the following generalization of Corollary 3.11 and Corollary 5.2.

COROLLARY 6.15. *Assume that $Q$ has padding.*

*(a) If $Q$ has an almost space optimal algorithm, then $Q$ has an almost (time) optimal algorithm.*
*(b) If $Q$ has an almost space optimal nondeterministic algorithm, then $Q$ has an almost (time) optimal algorithm.*

*Proof:* For (b) note that $\mathrm{LIST}(\mathrm{NL}, Q, \mathrm{NL})$ implies $\mathrm{LIST}(\mathrm{L}, Q, \mathrm{P})$ by Proposition 6.12 (a) and (b), and hence it implies $\mathrm{LIST}(\mathrm{P}, Q, \mathrm{P})$ by part (a) of the previous proposition. □

*The effective variant*. We start by introducing the effective notions.

DEFINITION 6.16. *Let $Q \subseteq \Sigma^*$.*

*(1) A proof system $\mathbb{P}$ for $Q$ is* effectively $p$-optimal *if there are two algorithms $\mathbb{T}$ and $\mathbb{B}$ such that (a) and (b) hold:*
  *(a) $\mathbb{T}$ and $\mathbb{B}$ compute functions defined on $\Sigma^*$; the values of $\mathbb{T}$ are in $\Sigma^*$, that of $\mathbb{B}$ in $\mathbb{N}[X]$;*
  *(b) for every proof system $\mathbb{P}'$ for $Q$*

$$\mathbb{T}(\mathbb{P}')$$

*is (the code of) a translation from $\mathbb{P}'$ into $\mathbb{P}$ such that for all $w' \in \Sigma^*$*

$$t_{\mathbb{T}(\mathbb{P}')}(w') \leq \mathbb{B}(\mathbb{P}')(t_{\mathbb{P}'}(w') + |w'|).$$

*(2) An effective listing of the* P*-subsets of $Q$ by* P*-machines is a listing $\mathbb{L}$ of the* P*-subsets of $Q$ by* P*-machines such that for some algorithms $\mathbb{I}$ and $\mathbb{B}$ we have (a) and (b):*
   *(a) $\mathbb{I}$ and $\mathbb{B}$ compute functions defined on $\Sigma^*$; the values of $\mathbb{I}$ are in $\mathbb{N}$, that of $\mathbb{B}$ in $\mathbb{N}[X]$;*
   *(b) for every polynomial time Turing machine $\mathbb{M}$ deciding a subset $Y$ of $Q$ the $\mathbb{I}(\mathbb{M})$th Turing machine $\mathbb{M}'$ listed by $\mathbb{L}$ decides $Y$ such that for all $x \in Y$*

$$t_{\mathbb{M}'}(x) \leq \mathbb{B}(\mathbb{M})(t_{\mathbb{M}}(x) + |x|).$$

Recall that in Definition 3.13 we introduced the notion of effectively almost optimal algorithm.

Standard proofs of Levin's Theorem (i.e., Theorem 6.6) implicitly show that in part (b) of it we can require that for every algorithm $\mathbb{I}$ inverting $f$ we get a polynomial $p \in \mathbb{N}_{d_0}[X]$ as stated there *effectively*. Now obvious changes of the proofs of Lemmas 6.3, 6.4, and 6.7 yield the following effective analogue of Theorem 6.2:

THEOREM 6.17. *For $Q$ with padding the following are equivalent:*

*(1) $Q$ has an effectively $p$-optimal proof system.*
*(2) $Q$ has an effectively almost optimal algorithm.*
*(3) $Q$ has an effective listing of the* P*-subsets of $Q$ by* P*-machines.*

Let us close this part by stating a corollary of this result obtained from Corollary 3.16 and Corollary 3.17. Part (b) is the "effective" generalization of a result due to Krajíček and Pudlák [Krajíček and Pudlák 1989].

COROLLARY 6.18.

*(a) If* $\mathrm{NP}[\mathrm{TC}] \not\subseteq \mathrm{P}[\mathrm{TC}^{\log \mathrm{TC}}]$*, then* TAUT *has no effectively $p$-optimal proof system.*
*(b) If* $\mathrm{NE} = \mathrm{E}$*, then* TAUT *has an effectively $p$-optimal proof system.*

### 6.3. A space version of Levin's result

THEOREM 6.19. *Let $\mathbb{F}$ be an algorithm computing a function $f : \Sigma^* \to \Sigma^*$. There exists an algorithm $\mathbb{O}$ such that:*

*(a) $\mathbb{O}$ inverts $f$ and $s_{\mathbb{O}}(y) = \infty$ for every input $y$, which is not in the range of $f$ (in particular, the algorithm $\mathbb{O}$ does not stop on $y$ not in the range);*
*(b) for every algorithm $\mathbb{I}$ inverting $f$ there is an $a \in \mathbb{N}$ such that for every $y$ in the range of $f$ we have*

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

*Proof:* First we introduce a notation. If $\mathbb{A}$ and $\mathbb{A}'$ are algorithms computing (partial) functions $g$ and $g'$ from $\Sigma^*$ to $\Sigma^*$, then by $\mathbb{A}; \mathbb{A}'$ we denote an algorithm that computes the function $g' \circ g$, i.e., $x \mapsto g'(g(x))$.

Now let $\mathbb{F}$ be an algorithm computing a function $f : \Sigma^* \to \Sigma^*$. Let $\mathbb{O}$ be the Turing machine that on input $y$ for $k = 0, 1, 2, \ldots$ and every Turing machine $\mathbb{M} \in \Sigma^*$ with $|\mathbb{M}; \mathbb{F}| \leq k$ simulates at most $|y| \cdot |\mathbb{M}; \mathbb{F}| \cdot (k - |\mathbb{M}; \mathbb{F}|) \cdot 2^{k-|\mathbb{M};\mathbb{F}|}$ steps of $\mathbb{M}; \mathbb{F}$ on input $y$ as long as space $\leq k - |\mathbb{M}; \mathbb{F}|$ is required;[4] if the simulation outputs $y$ (that is, $\mathbb{M}$ computes a string $w$ with $f(w) = y$), then it simulates $\mathbb{M}$ on $y$ (outputting the $w$) and halts.

[4] As $\ell := |\mathbb{M}; \mathbb{F}| \leq k$, the algorithm $\mathbb{M}; \mathbb{F}$ has at most $\ell$ states, so on input $y$ we have at most $|y| \cdot \ell \cdot (k - \ell) \cdot 2^{k-\ell}$ distinct configurations using space at most $k - |\mathbb{M}; \mathbb{F}|$. So, if $\mathbb{M}; \mathbb{F}$ on $y$ halts using at most this space, so will do the simulation.

Then, $\mathbb{O}$ inverts $f$ and $s_{\mathbb{O}}(y) = \infty$ for every input $y$, which is not in the range of $f$; hence, $\mathbb{O}$ satisfies (a). We turn to (b) and let $\mathbb{I}$ be any algorithm inverting $f$. There is $c \in \mathbb{N}$ such that the space required to simulate, given $\mathbb{I};\mathbb{F}$ and $y$, the algorithm $\mathbb{I};\mathbb{F}$ on input $y$ is less than or equal to

$$c + \log |\mathbb{I};\mathbb{F}| + \log |y| + s_{\mathbb{I};\mathbb{F}}(y). \tag{24}$$

We get an upper bound on the space that $\mathbb{O}$ requires on $y$ if we assume that $k$ gets a value such that $|\mathbb{I};\mathbb{F}| \leq k$ and the simulation of $\mathbb{I};\mathbb{F}$ on $y$ can be performed with space at most $k - |\mathbb{I};\mathbb{F}|$. Hence, by (24),

$$\begin{aligned} s_{\mathbb{O}}(y) &\leq O\big(|\mathbb{I};\mathbb{F}| + c + \log |\mathbb{I};\mathbb{F}| + \log |y| + s_{\mathbb{I};\mathbb{F}}(y)\big) \\ &\leq O\big(|\mathbb{I};\mathbb{F}| + c + \log |y| + s_{\mathbb{I}}(y) + \log |\mathbb{I}(y)| + s_{\mathbb{F}}(\mathbb{I}(y))\big) \\ &= O\big(\log |y| + s_{\mathbb{I}}(y) + \log |\mathbb{I}(y)| + s_{\mathbb{F}}(\mathbb{I}(y))\big). \end{aligned}$$

□

## 7. LOGICS AND LISTINGS

In the concepts "$L(\mathrm{C})$ is a $\mathrm{C}'$-bounded logic for C" and $\textsc{List}(\mathrm{C}, Q, \mathrm{C}')$ two complexity classes, C and $\mathrm{C}'$, appear. We show that they match, more precisely, we show:

THEOREM 7.1. *Let* $\mathrm{C} \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}\}$, $\mathrm{C}' \in \{\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}\}$, *and* $\mathrm{C} \subseteq \mathrm{C}'$. *Then*

$$L(\mathrm{C})_{\mathrm{inv}} \textit{ is a } \mathrm{C}'\textit{-bounded logic for } \mathrm{C} \iff \textsc{List}(\mathrm{C}, \textsc{Taut}, \mathrm{C}').$$

We remark that one can also define the notion of "almost $\mathrm{C}'$-optimal C-algorithm", an algorithm of type C almost optimal with respect to all $\mathrm{C}'$-algorithms, and analyze their relationship to the notions in the preceding result. For $\mathrm{C}, \mathrm{C}' \in \{\mathrm{P}, \mathrm{NP}\}$ and listings this has been done in [Sadowski 2002].

*Proof of Theorem 7.1*: We assume that $\mathrm{C} \subseteq \mathrm{C}'$ and that $L(\mathrm{C})_{\mathrm{inv}}$ is a $\mathrm{C}'$-bounded logic for C. Let PROP denote the class of all formulas of propositional logic. For a suitable vocabulary $\tau$ in logarithmic space we can associate with every $\alpha \in \textsc{Prop}$ a $\tau$-structure $\mathcal{A}(\alpha)$ such that

(i) every propositional variable $X$ of $\alpha$ corresponds to distinct elements $a_X, b_X$ of $\mathcal{A}(\alpha)$ and there is a unary $P \in \tau$ such that $P^{\mathcal{A}(\alpha)} = \{a_X \mid X \text{ variable of } \alpha\}$;
(ii) the class $\{\mathcal{B} \mid \mathcal{B} \cong \mathcal{A}(\alpha) \text{ for some } \alpha \in \textsc{Prop}\}$ of $\tau$-structures is axiomatizable by a $\mathrm{DTC}[\tau]$-sentence and therefore by an $L(\mathrm{C})[\tau]$-sentence $\varphi(\textsc{Prop})$;
(iii) if $\mathcal{B} \models \varphi(\textsc{Prop})$, then one can determine the unique $\alpha \in \textsc{Prop}$ with $\mathcal{B} \cong \mathcal{A}(\alpha)$ in logarithmic space.

An ordered $\tau_<$-structure of the form $(\mathcal{A}(\alpha), <)$ induces the assignment of the variables of $\alpha$ that sends a variable $X$ to TRUE if $a_X < b_X$. As in logarithmic space we can check whether this assignment satisfies $\alpha$, there is a $\mathrm{DTC}[\tau_<]$-sentence and hence an $L(\mathrm{C})[\tau_<]$-sentence $\varphi(\mathrm{sat})$ which for every $\alpha \in \textsc{Prop}$ expresses in $(\mathcal{A}(\alpha), <)$ that the assignment given by $<$ satisfies $\alpha$. We introduce the $L(\mathrm{C})[\tau_<]$-sentence

$$\varphi_0 := \big(\varphi(\textsc{Prop}) \to \varphi(\mathrm{sat})\big).$$

Then $\varphi_0$ is an $L(\mathrm{C})_{\mathrm{inv}}[\tau]$-sentence. Every assignment of $\alpha$ can be obtained by some ordering $<$ of $A(\alpha)$. Hence, by the definition of $\models_{L(\mathrm{C})_{\mathrm{inv}}}$, we see that for every $\alpha \in \textsc{Prop}$ and every $L(\mathrm{C})_{\mathrm{inv}}[\tau]$-sentence $\varphi$

$$\text{if } \mathcal{A}(\alpha) \models_{L(\mathrm{C})_{\mathrm{inv}}} (\varphi_0 \wedge \varphi), \text{ then } \alpha \in \textsc{Taut}. \tag{25}$$

For $\varphi \in L(\mathrm{C})_{\mathrm{inv}}[\tau]$ we consider the class of models of $(\varphi_0 \wedge \varphi)$, more precisely, the set

$$Q(\varphi) := \{\alpha \in \textsc{Prop} \mid \mathcal{A}(\alpha) \models_{L(\mathrm{C})_{\mathrm{inv}}} (\varphi_0 \wedge \varphi)\}.$$

*Claim*: The class of sets $Q(\varphi)$, where $\varphi$ ranges over all $L(\mathrm{C})_{\mathrm{inv}}$-sentences coincides with the class of C-subsets of TAUT.

*Proof of the Claim*: First let $Q$ be a C-subset of TAUT. If $Q$ is finite, it is easy to see that $Q = Q(\varphi)$ for some $\varphi \in L(\mathrm{C})_{\mathrm{inv}}$. Now let $Q$ be infinite. The class $\{\mathcal{B} \mid \mathcal{B} \cong \mathcal{A}(\alpha) \text{ for some } \alpha \in Q\}$ is in C (by (ii) and (iii) as $\mathrm{L} \subseteq \mathrm{C}$). As we assume that $L(\mathrm{C})_{\mathrm{inv}}$ is a logic for C, this class is axiomatizable by an $L(\mathrm{C})_{\mathrm{inv}}[\tau]$-sentence $\varphi$. As the class contains arbitrarily large structures, the formula $\varphi$ is invariant. We show that $Q = Q(\varphi)$.

Assume first that $\alpha \in Q(\varphi)$, i.e., $\mathcal{A}(\alpha) \models_{L(\mathrm{C})_{\mathrm{inv}}} (\varphi_0 \wedge \varphi)$. Then, by invariance of $\varphi$, we have $\mathcal{A}(\alpha) \models_{L(\mathrm{C})_{\mathrm{inv}}} \varphi$ and thus $\alpha \in Q$. Conversely, assume that $\alpha \in Q$. Then $\mathcal{A}(\alpha) \models_{L(\mathrm{C})_{\mathrm{inv}}} \varphi$. As $\alpha \in$ TAUT, in order to get $\mathcal{A}(\alpha) \models_{L(\mathrm{C})_{\mathrm{inv}}} (\varphi_0 \wedge \varphi)$ (and hence, $\alpha \in Q(\varphi)$), it suffices to show that $(\varphi_0 \wedge \varphi)$ is $\leq |A(\alpha)|$-invariant. So let $\mathcal{B}$ be a $\tau$-structure with $|B| \leq |A(\alpha)|$. If $\mathcal{B} \not\models_{L(\mathrm{C})_{\mathrm{inv}}} \varphi$, then, by invariance of $\varphi$, we have $(\mathcal{B}, <^B) \not\models_{L(\mathrm{C})} (\varphi_0 \wedge \varphi)$ for all orderings $<^B$ on $B$; if $\mathcal{B} \models_{L(\mathrm{C})_{\mathrm{inv}}} \varphi$, then $\mathcal{B} \cong \mathcal{A}(\beta)$ for some $\beta \in Q \subseteq$ TAUT. Hence, $(\mathcal{B}, <^B) \models_{L(\mathrm{C})} (\varphi_0 \wedge \varphi)$ for all orderings $<^B$ on $B$.

We still have to show that $Q(\varphi)$ is a C-subset of TAUT for every $\varphi \in L(\mathrm{C})_{\mathrm{inv}}[\tau]$. So we fix $\varphi \in L(\mathrm{C})_{\mathrm{inv}}[\tau]$. By (25), $Q(\varphi) \subseteq$ TAUT. As $L(\mathrm{C})_{\mathrm{inv}}$ is a logic for C, we have $Q(\varphi) \in \mathrm{C}$. $\dashv$

As $L(\mathrm{C})_{\mathrm{inv}}$ is a $\mathrm{C}'$-bounded logic for C, the corresponding algorithm $\mathbb{A}$ for the satisfaction relation (cf. Definition 4.2 (b)) restricted to $(\varphi_0 \wedge \varphi)$ with $\varphi \in L(\mathrm{C})_{\mathrm{inv}}[\tau]$ yields an algorithm of type $\mathrm{C}'$ accepting $Q(\varphi)$. Thus, by the Claim, the classes $Q(\varphi)$ where $\varphi$ ranges over all $L(\mathrm{C})_{\mathrm{inv}}[\tau]$-sentences witness that $\mathrm{LIST}(\mathrm{C}, \mathrm{TAUT}, \mathrm{C}')$.

Now let us assume that $\mathrm{LIST}(\mathrm{C}, \mathrm{TAUT}, \mathrm{C}')$. It suffices to show that $p\text{-HALT}_> \in \mathrm{XC}'_{\mathrm{uni}}$ as then Theorem 4.1 yields the claim. Since $\mathrm{L} \subseteq \mathrm{C}$, we have $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{C}')$ (by Proposition 6.12 (b)). First assume that $\mathrm{C}'$ is a space complexity class. If $\mathrm{C}' = \mathrm{L}$, then $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{L})$ and therefore TAUT has an almost space optimal algorithm (by Theorem 6.11) and hence $p\text{-HALT}_> \in \mathrm{XL}_{\mathrm{uni}}$ (by Theorem 3.10).

Let $\mathrm{C}' = \mathrm{NL}$. We fix a logspace one-to-one reduction $\langle \mathbb{M}, 1^n \rangle \mapsto \alpha(\mathbb{M}, 1^n)$ from $\mathrm{HALT}_>$ (the classical problem underlying $p\text{-HALT}_>$) to TAUT, which is logspace invertible. Furthermore, let $\mathbb{B}$ be an algorithm that on input $\mathbb{M}$, a nondeterministic Turing machine, computes $t_{\mathbb{M}}(\lambda)$, the least $k$ such that there is an accepting run of $\mathbb{M}$ on the empty string $\lambda$, by "brute force." Let $\mathbb{L}$ be a listing witnessing $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{NL})$. We show that the following nondeterministic algorithm $\mathbb{A}$ witnesses that $p\text{-HALT}_> \in \mathrm{XNL}_{\mathrm{uni}}$:

```
A   // M a nondeterministic Turing machine, 1^n with n ∈ N
1.  guess x ∈ {B, L}
2.  if x = B then simulate B on M
3.          if B halts then
4.                  if n < t_M(λ) then accept
5.  if x = L then guess i ≥ 1
6.          simulate L till it outputs the ith machine, say, C
7.          simulate C on α(M, 1^n)
8.          if C accepts then accept.
```

Whenever during the simulation of $\mathbb{C}$ on $\alpha(\mathbb{M}, 1^n)$ a bit of the input $\alpha(\mathbb{M}, 1^n)$ is required, the algorithm $\mathbb{A}$ simulates the reduction $\langle \mathbb{M}, 1^n \rangle \mapsto \alpha(\mathbb{M}, 1^n)$ till this bit is obtained.

Clearly $\mathbb{A}$ accepts $\mathrm{HALT}_>$. We still have to verify that for some function $f : \mathbb{N} \to \mathbb{N}$ and every $\langle \mathbb{M}, 1^n \rangle \in p\text{-HALT}_>$ we have $s_{\mathbb{A}}(\langle \mathbb{M}, 1^n \rangle) \leq f(|\mathbb{M}|) \cdot \log n$. We fix $\langle \mathbb{M}, 1^n \rangle \in p\text{-HALT}_>$ and consider the following two cases:

*Case "*$\langle \mathbb{M}, 1^\ell \rangle \notin p$-HALT$_>$ *for some* $\ell \in \mathbb{N}$*"*: Then the run of $\mathbb{A}$ on input $\langle \mathbb{M}, 1^n \rangle$, where $x = \mathbb{B}$ is chosen in Line 1, witnesses that the space $s_{\mathbb{A}}(\langle \mathbb{M}, 1^n \rangle)$ can be bounded by $c_{\mathbb{M}} \cdot \log |n|$ for some constant $c_{\mathbb{M}} \in \mathbb{N}$.

*Case "*$\langle \mathbb{M}, 1^\ell \rangle \in p$-HALT$_>$ *for all* $\ell \in \mathbb{N}$*"*: Then $\left\{ \alpha(\mathbb{M}, 1^\ell) \;\middle|\; \ell \in \mathbb{N} \right\}$ is an L-subset of TAUT and hence $\mathbb{L}$ lists a machine $\mathbb{C}$ accepting this set. As $\mathbb{C}$ is logspace, the claim follows immediately.

The cases where C$'$ is a time complexity class are treated similarly using algorithms analogous to those used for time classes in Section 3. □

We already mentioned that we do not know any direct proof of the following result; in particular, we do not know whether we can replace TAUT by any $Q$ with padding.

COROLLARY 7.2. *If* LIST(L, TAUT, L)*, then* LIST(NL, TAUT, NL).

## REFERENCES


CHANDRA, A. AND HAREL, D. 1982. Structure and complexity of relational queries. *Journal of Computer and System Sciences 25,* 1, 99–128.

CHEN, Y. AND FLUM, J. 2010a. A logic for PTIME and a parameterized halting problem. In *Fileds of Logic and Computation*. Springer, 251–276.

CHEN, Y. AND FLUM, J. 2010b. 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. Springer, 321–332.

CHEN, Y. AND FLUM, J. 2010c. On slicewise monotone parameterized problems and optimal proof systems for TAUT. In *Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL'10)*. Lecture Notes in Computer Science 6247. Springer, 200–214.

CHEN, Y. AND FLUM, J. 2010d. On the complexity of Gödel's proof predicate. *The Journal of Symbolic Logic 75,* 1, 239–254.

CHEN, Y. AND FLUM, J. 2011. Listings and logics. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS'11)*. IEEE Computer Society, 165–174.

COOK, S. AND RECKHOW, R. 1979. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic 44,* 1, 36–50.

DOWNEY, R. AND FELLOWS, M. 1999. *Parameterized Complexity*. Springer-Verlag.

FAGIN, R. 1974. Generalized first–order spectra and polynomial–time recognizable sets. In *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, R. M. Karp, Ed. 43–73.

FLUM, J. AND GROHE, M. 2006. *Parameterized Complexity Theory*. Springer.

GROHE, M. 2009. Fixed-point definability and polynomial time. In *Proceedings of the 18th EACSL Annual Conference on Computer Science Logic (CSL'09)*. Lecture Notes in Computer Science 5771. Springer, 20–23.

GUREVICH, Y. 1988. Logic and the challenge of computer science. In *Current Trends in Theoretical Computer Science*. Computer Science Press, 1–57.

HARTMANIS, J. AND HEMACHANDRA, L. 1988. Complexity classes without machines: On complete languages for up. *Theoretical Computer Science 25*, 129–142.

HITCHCOCK, J. AND PAVAN, A. 2004. Hardness hypotheses, derandomization, and circuit complexity. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*. Lecture Notes in Computer Science 3328. Springer, 336–347.

IMMERMAN, N. 1986. Relational queries computable in polynomial time. *Information and Control 68*, 86–104.

IMMERMAN, N. 1987. Languages that capture complexity classes. *SIAM Journal on Computing 16,* 4, 770–778.

IMMERMAN, N. 1988. Nondeterministic space is closed under complement. *SIAM Journal on Computing 17,* 5, 935–938.

KÖBLER, J., MESSNER, J., AND TORÁN, J. 2003. Optimal proof systems imply complete sets for promise classes. *Information and Computation 184,* 1, 71–92.

KOWALCZYK, K. 1984. Some connections between presentability of complexity classes and the power of formal systems of reasonning. In *Proceedings of Mathematical Foundations of Computer Science 1984 (MFCS'84)*. Lecture Notes in Computer Science 176. Springer, 364–369.

KRAJÍČEK, J. AND PUDLÁK, P. 1989. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic 54,* 3, 1063–1079.

LEVIN, L. 1973. Universal search problems. *Problems of Information Transmission 9,* 3, 265–266.

MESSNER, J. 2000. On the simulation order of proof systems. Ph.D. thesis, University of Erlangen.

MESSNER, J. AND TORÁN, J. 1998. Optimal proof systems for propositional logic and complete sets. In *Proceedings of the 15th Annual Symposium of Theoretical Aspects of Computer Science (STACS'98)*. Lecture Notes in Computer Science 1373. Springer, 477–487.

NASH, A., REMMEL, J., AND VIANU, V. 2005. PTIME queries revisited. In *Proceedings of the 10th International Conference on Database Theory (ICDT'05)*. Lecture Notes in Computer Science 3363. Springer, 274–288.

SADOWSKI, Z. 2002. On an optimal propositional proof system and the structure of easy subsets. *Theoretical Computer Science 288,* 1, 181–193.

VARDI, M. Y. 1982. The complexity of relational query languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC'82)*. 137–146.