Electronic Colloquium on Computational Complexity, Report No. 20 (2011)

Listings and logics

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

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

December 20, 2010

Abstract

There are standard logics DTC, TC, and LFP capturing the complexity classes L, NL, and P on ordered structures, respectively. In [5] we have shown that $\mathrm{LFP}_{\mathrm{inv}}$, the "order-invariant least fixed-point logic LFP," captures P (on *all* finite structures) if and only if there is a listing of the P-subsets of the set TAUT of propositional tautologies. By a thorough analysis of this relationship between listings and logics we are able to extend the result to listings of the L-subsets (NL-subsets) of TAUT and the logic $\mathrm{DTC}_{\mathrm{inv}}$ ($\mathrm{TC}_{\mathrm{inv}}$). As a byproduct we get that $\mathrm{LFP}_{\mathrm{inv}}$ captures P if $\mathrm{DTC}_{\mathrm{inv}}$ captures L.

Furthermore, we show that the existence of a listing of the L-subsets of TAUT is equivalent to the existence of an almost space optimal algorithm for TAUT. To obtain this result we have to derive a space version of a theorem of Levin on optimal inverters.

1. Introduction

It is well-known that for standard complexity classes C (as L, NL and P) the existence of a logic capturing C is equivalent to the existence of a listing (or effective enumeration) of the classes of structures closed under isomorphism in C by means of Turing machines of type C that decide them (or equivalently, to such a listing of the classes of graphs closed under isomorphism). Recently [5] we have shown for C = P that such a listing exists if there is a listing of the P-subsets of the set TAUT of propositional tautologies; more explicitly, a listing of the subsets in P of TAUT by means of polynomial time Turing machines deciding them. Even more, it is shown in [5] that the following two statements are equivalent:

(i) There is a listing of the P-subsets of TAUT.

(ii) The logic $\mathrm{LFP}_{\mathrm{inv}}$, the "order-invariant least fixed-point logic LFP," [1] captures P.

In the course of the proof a further statement (in [15] shown to be) equivalent to (ii) played a prominent role:

(iii) There is an algorithm deciding for every nondeterministic Turing machine $\mathbb{M}$ and every natural number n whether $\mathbb{M}$ accepts the empty input tape in $\leq n$ steps and which for fixed $\mathbb{M}$ runs in time polynomial in n.

The starting point for the investigations which led to this paper were an observation and a question:

The observation. One easily verifies (cf. Proposition 2.3) that one gets a listing of the P-subsets of TAUT if one assumes that there is a listing of its L-subsets. In particular, then $\mathrm{LFP}_{\mathrm{inv}}$ captures P. Thus, we asked ourselves whether then we even get that $\mathrm{DTC}_{\mathrm{inv}}$, the "order-invariant deterministic transitive closure logic DTC," [1] captures L.

The question. It is known that E = NE implies (i)–(iii) are true. Nevertheless, as already done in [7, 13] we also conjecture that (i)–(iii) are false (under reasonable complexity-theoretic assumptions); in particular,

[1] It is well-known that LFP captures P on *ordered* structures and that deterministic transitive closure logic DTC captures L on ordered structures.

ISSN 1433-8092

there is *no* algorithm deciding the acceptance problem for Turing machines mentioned in (iii), which for fixed $\mathbb{M}$ runs in time polynomial in n. As we were not able to prove this conjecture, we asked ourselves whether assuming, say, P $\neq$ L, we can at least show that there is no algorithm deciding this acceptance problem which for fixed $\mathbb{M}$ runs in space logarithmic in n.

At the end we realized that the equivalence of (i), (ii), and (iii) lifts to their L-analogues and to their NL-analogues.

For complexity classes C and C$'$ we consider listings of the C-subsets of TAUT by means of Turing machines of type C$'$; we write LIST(C, TAUT, C$'$) if such a listing exists. For the classes P and NP such listings were already considered and put to good use by Sadowski in [16]. This more general notion is also meaningful in the context of logics (for P and NP already remarked in [4]); loosely speaking, if LIST(C, TAUT, C$'$), then in the order-invariant logic corresponding to C we can axiomatize the classes of structures in C and we can show that there is an algorithm solving its satisfaction relation of type C$'$. By this general approach we get further new insights (cf. Corollary 3.9), among others, we show:

> If there is an algorithm solving the satisfaction relation for DTC_{inv}, which for fixed DTC_{inv}-sentence runs in polynomial time, then LFP_{inv} captures P.

In particular, LFP_{inv} captures P if DTC_{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.

The relationship between listings and logics is also fruitful for the side of the listings (cf. Corollary 3.10), e.g., we get

$$\text{If } \text{LIST}(\text{L}, \text{TAUT}, \text{L}), \text{ then } \text{LIST}(\text{NL}, \text{TAUT}, \text{NL}). \tag{1}$$

As already indicated the concept of a listing of subsets of TAUT is also relevant in the context of propositional proof systems. In fact, the existence of a listing of P-subsets of TAUT is equivalent to the existence of a p-optimal propositional proof system (cf. [16, 2]) and hence to the existence of an almost optimal algorithm for TAUT (cf. [13]). We show the analogues for L in Section 4. There we introduce the concepts of space optimal logspace proof system and of almost space optimal algorithm for arbitrary problems Q. For Q with padding it turns out that they exist if and only if there is a listing of the L-subsets of Q by L-machines. In particular, by (1), 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.

The content of the different sections is the following. We start (Section 2) by introducing the concept of listing for arbitrary problems Q (and not only for TAUT) and by studying the relationship between different types of listings. In Section 3, we analyze the connection between listings and logics and prove a general result (cf. Theorem 3.8) which yields the different results concerning listings and logics mentioned above. In Section 4 we derive the space versions of results relating listings with optimal proof systems and almost optimal algorithms. To prove them, (more or less explicitly) we need a space version of a result of Levin [14] on the existence of optimal inverters (cf. Theorem 4.10). We have deferred its proof to the last section. As for listings, also for optimal proof system and almost optimal algorithms there are variants of our result taking into account two complexity classes. In order not to get lost in generalizations we do not get into this matter.

We have organized the material in such a way that a reader only interested in the relationship between listings and optimal proof system and almost optimal algorithms may skip Section 3.

2. Listings

In this section we introduce the notion of listing and derive some of their basic properties. We start by recalling some notions from complexity theory and fixing our notation.

2.1. Some Preliminaries. We denote the alphabet $\{0, 1\}$ by Σ. The length of a string $x \in \Sigma^*$ is denoted by $|x|$. We identify problems with subsets Q of Σ^*. 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. By NP and NL we denote the corresponding nondeterministic classes. Let C be a complexity class and Q a problem. We say that X is a C-*subset of* Q if $X \subseteq Q$ and $X \in \mathrm{C}$.

A problem $Q \subseteq \Sigma^*$ *has padding* if there is a function *pad* : $\Sigma^* \times \Sigma^* \to \Sigma^*$ having the following properties:

(i) It is computable in logarithmic space.

(ii) For any $x, y \in \Sigma^*$, $\mathit{pad}(x, y) \in Q$ if and only if $x \in Q$.

(iii) For any $x, y \in \Sigma^*$, $|\mathit{pad}(x, y)| > |x| + |y|$.

(iv) There is a logspace algorithm which, given $\mathit{pad}(x, y)$ recovers y.

By $\langle \ldots, \ldots, \ldots \rangle$ we denote some standard logspace computable tupling functions with logspace computable inverses.

All deterministic and nondeterministic Turing machines have Σ as their alphabet. If necessary we will not distinguish between an algorithm and a Turing machine and also between a Turing machine and its code, a string in Σ^*. If $\mathbb{M}$ is a Turing machine, we denote by $|\mathbb{M}|$ the length of its code. If $\mathbb{M}$ is a deterministic or nondeterministic Turing machine, then $L(\mathbb{M})$ denotes the language accepted by $\mathbb{M}$.

By PROP and TAUT we denote the class of all propositional formulas and the class of propositional formulas that are tautologies.

In this paper, C, C′, C_0, ... will always denote one of the complexity classes L, P, NL or NP.

2.2. Listings. Effective enumerations of problems by means of Turing machines have been used to characterize promise classes possessing complete languages (e.g., see [8, 12]). In the context of optimal proof systems, listings of subsets of TAUT have been used systematically by Sadowski (see [16]).

Definition 2.1. Let C and C′ be complexity classes. For $Q \subseteq \Sigma^*$ a *listing of the* C-*subsets of* Q *by* C′-*machines* is an algorithm $\mathbb{L}$ that, once having been started, eventually yields as outputs Turing machines of type C′

$$\mathbb{M}_1, \mathbb{M}_2, \ldots$$

such that

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

Often we speak of the listing $\mathbb{M}_1, \mathbb{M}_2, \ldots$ (instead of the listing $\mathbb{L}$). We write $\textsc{List}(\mathrm{C}, Q, \mathrm{C}')$ if there is a listing of the C-subsets of Q by C′-machines.

By systematically adding polynomial time clocks (if C′ is a time class) or devices controlling the space used, if necessary we may assume that *all* runs of the machines $\mathbb{M}_i$ on *any* input satisfy the time or space bound characteristic for C′.

For $Q =$ TAUT part (c) of the next proposition has already been shown in [16] by completely different means.

Proposition 2.2. *(a) Let* C, C′, *and* C″ *be complexity classes with* $\mathrm{C}' \subseteq \mathrm{C}''$. *If* $\textsc{List}(\mathrm{C}, Q, \mathrm{C}')$, *then* $\textsc{List}(\mathrm{C}, Q, \mathrm{C}'')$.

(b) Let C, C′, *and* C_0 *be complexity classes with* $\mathrm{C}_0 \subseteq \mathrm{C} \subseteq \mathrm{C}'$. *If* $\textsc{List}(\mathrm{C}, Q, \mathrm{C}')$, *then* $\textsc{List}(\mathrm{C}_0, Q, \mathrm{C}')$.

(c) Assume Q *has padding. Then*

$$\textsc{List}(\mathrm{NP}, Q, \mathrm{NP}) \iff \textsc{List}(\mathrm{P}, Q, \mathrm{NP}).$$

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

Proof: (a) is trivial. (b) Let $\mathbb{M}'$ be a Turing machine of type C′ and $\mathbb{M}_0$ one of type 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

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

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 C$_0$, then the listing $(\mathbb{M}'_i(\mathbb{M}_j))_{i\geq 1,\ j\geq 1}$ witnesses that $\textsc{List}(\mathrm{C}_0, Q, \mathrm{C}')$.

(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 $\textsc{List}(\mathrm{P}, Q, \mathrm{NP})$. For a nondeterministic Turing machine $\mathbb{M}$, we set

$$\mathit{Comp}(\mathbb{M}) := \big\{\mathit{pad}(x, c) \mid x \in \Sigma^* \text{ and } c \text{ is a computation } [2] \text{ of } \mathbb{M} \text{ accepting } x\big\}.$$

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

$$\mathit{Comp}(\mathbb{M}) \subseteq Q \iff L(\mathbb{M}) \subseteq Q. \tag{2}$$

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 $\mathit{pad}(x, c)$. □

For the set $\mathit{Comp}(\mathbb{M})$ introduced for any nondeterministic Turing machine in the previous proof of (c), we even have $\mathit{Comp}(\mathbb{M}) \in \mathrm{L}$. We use this to obtain the following observation that, as already mentioned in the Introduction, was one of our starting points.

Proposition 2.3. *(a) If Q has padding and* $\textsc{List}(\mathrm{L}, Q, \mathrm{L})$*, then* $\textsc{List}(\mathrm{P}, Q, \mathrm{P})$.

(b) If Q has padding and $\textsc{List}(\mathrm{NL}, Q, \mathrm{NL})$*, then* $\textsc{List}(\mathrm{NP}, Q, \mathrm{NP})$.

Proof: (a) For deterministic Turing machines $\mathbb{D}$ and $\mathbb{D}'$ let $\mathbb{D}(\mathbb{D}')$ be the deterministic machine that on input x, first by simulating $\mathbb{D}'$ on input x stores its computation c and then runs $\mathbb{D}$ on input $\mathit{pad}(x, c)$. By (2), if $\mathbb{D}_1, \mathbb{D}_2, \ldots$ is a listing of the L-subsets of Q by L-machines and $\mathbb{D}'_1, \mathbb{D}'_2, \ldots$ an enumeration of all deterministic polynomial time Turing machines, then the enumeration $(\mathbb{D}_i(\mathbb{D}'_j))_{i\geq 1,\ j\geq 1}$ witnesses that $\textsc{List}(\mathrm{P}, Q, \mathrm{P})$.

The proof of (b) is obtained by obvious modifications. □

We were unable to prove the logspace analogue of Proposition 2.2(c), however as a byproduct of our main result relating listings and logics (Theorem 3.8) we get that (c) holds for $Q = \textsc{Taut}$, that is, $\textsc{List}(\mathrm{L}, \textsc{Taut}, \mathrm{NL})$ implies $\textsc{List}(\mathrm{NL}, \textsc{Taut}, \mathrm{NL})$.

3. Listings and logics

In [5] we have shown that the logic $\mathrm{LFP}_{\mathrm{inv}}$ is a P-bounded logic for P if and only if $\textsc{List}(\mathrm{P}, \textsc{Taut}, \mathrm{P})$. Here, among others, we show the L-analogue and the NL-analogue of this result. We have remind the reader only interested in the relationship between listings and optimal proof system and almost optimal algorithms that he may skip this section.

3.1. Some preliminaries. We start by recalling some concepts from logic and complexity.

Structures. A *vocabulary* τ is a finite set of relation symbols. Each relation symbol has an *arity*. A *structure* $\mathcal{A}$ of vocabulary τ, or *τ-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.*

For a structure $\mathcal{A}$ we denote by $\|\mathcal{A}\|$ the size of $\mathcal{A}$, that is, the length of a reasonable encoding of $\mathcal{A}$ as a string in Σ^*. We only consider properties of structures that are invariant under isomorphisms, so it suffices that from the encoding of $\mathcal{A}$ we can recover $\mathcal{A}$ up to isomorphism.

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

Logics. For our purposes a *logic* L consists

- for every vocabulary τ of a set $L[\tau]$ of strings, the set of L*-sentences of vocabulary* τ and of an algorithm that for every vocabulary τ and every string ξ decides whether $\xi \in L[\tau]$ (in particular, $L[\tau]$ is decidable for every τ);
- of a *satisfaction relation* $\models_L$; if $(\mathcal{A}, \varphi) \in \models_L$, written $\mathcal{A} \models_L \varphi$, then $\mathcal{A}$ is a τ-structure and $\varphi \in L[\tau]$ for some vocabulary τ; furthermore for each $\varphi \in L[\tau]$ the class $\mathrm{Mod}_L(\varphi) := \{\mathcal{A} \mid \mathcal{A} \models_L \varphi\}$ of *models of* φ is closed under isomorphism.

Recall that C and C$'$ always range over the complexity classes L, P, NL, and NP.

Definition 3.1. Let L be a logic and C a complexity class.

(a) L *is a logic for* C if for all vocabularies τ and all classes D (of encodings) of τ-structures closed under isomorphism we have

$$D \in \mathrm{C} \iff D = \mathrm{Mod}_L(\varphi) \text{ for some } \varphi \in L[\tau].$$

(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 a deterministic (nondeterministic) algorithm $\mathbb{A}$ deciding (accepting) $\models_L$, which for fixed φ witnesses that $\mathrm{Mod}_L(\varphi) \in \mathrm{C}'$.

Clearly, if L is a C$'$-bounded logic for C, then $\mathrm{C} \subseteq \mathrm{C}'$. If $\mathrm{C}' = \mathrm{C}$, 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 C$' =$ C. In fact, one expects that L can be viewed as a higher programming language for C, that is, that for fixed L-sentence φ the algorithm $\mathbb{A}$ in (b) witnesses that $\mathrm{Mod}_L(\varphi) \in \mathrm{C}$.

However we use this more liberal, a little bit artificial notion as in this way we obtain some nontrivial consequences from the main theorem (see Corollary 3.9(d)).

It is well-known that there are NP-bounded logics for NP while for C $=$ L, C $=$ P, and C $=$ NL it is open whether there is a C-bounded logic for C.

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

We say that a logic L is a C$'$*-bounded logic for* C *on ordered structures* if (a) and (b) of the previous definition hold for classes D of *ordered* structures and for *ordered* structures $\mathcal{A}$, respectively. In (b), for fixed $\varphi \in L[\tau_<]$ the algorithm $\mathbb{A}$ must witness that the class of ordered models of φ is in C$'$.

We denote by FO, DTC, TC, and LFP *first-order logic, deterministic transitive closure logic, transitive closure logic*, and *least fixed-point logic*, respectively. Essentially we only need the following properties of these logics due to Immerman [10, 11] and the last one to Immerman [9] and Vardi [17]

Theorem 3.2. *(1)* DTC *is an* L*-bounded logic for* L *on ordered structures.*

(2) TC *is an* NL*-bounded logic for* NL *on ordered structures.*

(3) LFP *is a* P*-bounded logic for* P *on ordered structures.*

If C is L, NL, or P, then we let $L(\mathrm{C})$ be the logic DTC, TC, and LFP, respectively.

Invariant sentences and the logic $L_{\mathbf{inv}}$. We start by introducing the notion of (order-)invariant sentence.

Definition 3.3. Let L be a logic.

- Let φ be an $L[\tau_<]$-sentence and $n \geq 1$. We say that φ is $\leq n$*-invariant* if for all τ-structures $\mathcal{A}$ with $|A| \leq n$ and all orderings $<_1$ and $<_2$ on A we have

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

– L-INV:= $\big\{(\varphi, n) \mid \varphi\ L\text{-sentence},\ n \geq 1 \text{ and } \varphi \leq n\text{-invariant}\big\}$.

Now we define the logic L_{inv}. For every vocabulary τ we set

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

and we define the satisfaction relation by

$$\mathcal{A} \models_{L_{\text{inv}}} \varphi \iff \Big((\varphi, \leq |A|) \in L\text{-INV and } (\mathcal{A}, <) \models_L \varphi \text{ for some ordering } < \text{ on } A\Big). \tag{3}$$

Assume that for every L-sentence φ, say, of vocabulary τ there is a sentence $\neg\varphi$ of the same vocabulary such that $\text{MOD}(\neg\varphi) = \big\{\mathcal{A} \mid \mathcal{A}\ \tau\text{-structure and } \mathcal{A} \notin \text{MOD}(\varphi)\big\}$. As

$$(\varphi, \leq n) \in L\text{-INV} \iff (\neg\varphi, \leq n) \in L\text{-INV},$$

we get for every structure $\mathcal{A}$

$$(\varphi, \leq |A|) \in L\text{-INV} \iff (\mathcal{A} \models_{L_{\text{inv}}} \varphi \text{ or } \mathcal{A} \models_{L_{\text{inv}}} \neg\varphi). \tag{4}$$

Using Theorem 3.2 it is easy to see:

Proposition 3.4. *If* C *is one of the classes* L, NL*, and* P*, then* $L(\text{C})_{\text{inv}}$ *is a logic for* C.

Some further complexity classes. We will consider the complexity of the following *acceptance problem for nondeterministic Turing machines*:

$\text{ACC}_\leq$	
Instance:	A nondeterministic Turing machine $\mathbb{M}$ and $n \in \mathbb{N}$ in unary.
Question:	Does $\mathbb{M}$ accept the empty input tape in $\leq n$ steps?

Hence, $\text{ACC}_\leq \subseteq \Sigma^* \times \mathbb{N}$. Similarly, we have L-INV $\subseteq \Sigma^* \times \mathbb{N}$ for any logic L. In the following definition we assume that the natural numbers, the second components of the input instances, are given in unary.

Definition 3.5. Let C be a deterministic (nondeterministic) complexity class. A problem $R \subseteq \Sigma^* \times \mathbb{N}$ is in the class XC_{uni} if there is a deterministic (nondeterministic) algorithm deciding (accepting) R and witnessing for every $k \in \mathbb{N}$ that the problem

$$R_k := \big\{(x, n) \in R \mid |x| = k\big\},$$

is in C. For example, R is in the class XL_{uni} if there is a deterministic algorithm $\mathbb{A}$ deciding R and requiring for $(x, n) \in R$ space at most $f(|x|) \cdot \log n$ for some function $f : \mathbb{N} \to \mathbb{N}$. And R is in the class XNL_{uni} if there is a nondeterministic algorithm $\mathbb{A}$ accepting R such that for some function $f : \mathbb{N} \to \mathbb{N}$ and every $(x, n) \in R$ there is an accepting run requiring space at most $f(|x|)\cdot\log n$. Finally, the problem $R \subseteq \Sigma^* \times \mathbb{N}$ is in the class co-XC_{uni} if its complement $(\Sigma^* \times \mathbb{N}) \setminus R$ is in XC_{uni}. [3]

It is not hard to adapt the proof of the corresponding reduction in Theorem 1 of [3] showing the following result due to [15]

$$\text{if } \text{ACC}_\leq \in \text{co-XP}_{\text{uni}}, \text{ then LFP-INV} \in \text{XP}_{\text{uni}} \tag{5}$$

in order to get

$$\text{if } \text{ACC}_\leq \in \text{co-XNP}_{\text{uni}}, \text{ then LFP-INV} \in \text{XNP}_{\text{uni}} \tag{6}$$

and to get the following result generalizing (5):

Proposition 3.6. *Let* C *be one of the classes* L, NL*, or* P*. If* $\text{ACC}_\leq \in \text{co-XC}_{\text{uni}}$*, then* $L(\text{C})$-INV $\in \text{XC}_{\text{uni}}$.

[3] These are classes of so-called uniform parameterized complexity theory. In our type of problems $R \subseteq \Sigma^* \times \mathbb{N}$ the parameter of an instance (x, n) of R is $|x|$.

We will use the following lemma.

Lemma 3.7. *Let* C *be one of the classes* L*,* NL*, or* P *and* $\mathrm{C} \subseteq \mathrm{C}'$*. The following statements are equivalent:*

(i) $L(\mathrm{C})_{\mathrm{inv}}$ *is a* C'*-bounded logic for* C*.*

(ii) $L(\mathrm{C})$-INV $\in \mathrm{XC}'_{\mathrm{inv}}$.

Proof: By Proposition 3.4, we already know that $L(\mathrm{C})_{\mathrm{inv}}$ is a logic for C, that is, condition (a) in Definition 3.1 is fulfilled. By (3) and (4), the second condition of this definition is fulfilled if and only if $L(\mathrm{C})$-INV $\in \mathrm{XC}'_{\mathrm{inv}}$. □

3.2. Logics and listings. We start by stating the main result of this section.

Theorem 3.8. *Let* C *be one of the classes* L*,* NL*, or* P *and* $\mathrm{C} \subseteq \mathrm{C}'$*.*

(a) If $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{C})$*, then* $L(\mathrm{C})_{\mathrm{inv}}$ *is a* C*-bounded logic for* C*.*

(b) Let $\mathrm{C} \subseteq \mathrm{C}'$*. If* $L(\mathrm{C})_{\mathrm{inv}}$ *is a* C'*-bounded logic for* C*, then* $\mathrm{LIST}(\mathrm{C}, \mathrm{TAUT}, \mathrm{C}')$*.*

The strength of this result is indicated by the following corollaries, where we list some consequences:

Corollary 3.9. *(a)* $\mathrm{DTC}_{\mathrm{inv}}$ *is an* L*-bounded logic for* L *if and only if* $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{L})$*.*

(b) $\mathrm{TC}_{\mathrm{inv}}$ *is an* NL*-bounded logic for* NL *if and only if* $\mathrm{LIST}(\mathrm{NL}, \mathrm{TAUT}, \mathrm{NL})$*.*

(c) 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*.*

(d) 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)–(b) are immediate consequences of Theorem 3.8 and of the fact that $\mathrm{LIST}(\mathrm{NL}, \mathrm{TAUT}, \mathrm{NL})$ implies $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{NL})$ (by Proposition 2.2(b)).

(c) If $\mathrm{DTC}_{\mathrm{inv}}$ is an L-bounded logic for L, then $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{L})$ and hence, $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{NL})$ and $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{P})$ (by Proposition 2.2(a)). Now the claims follow from Theorem 3.8(a).

(d) If $\mathrm{DTC}_{\mathrm{inv}}$ is a P-bounded logic for L, then $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{P})$ (by Theorem 3.8(b)) and therefore, $\mathrm{LFP}_{\mathrm{inv}}$ is a P-bounded logic for P by Theorem 3.8(a). □

And also for the listings, we get new insights, for example:

Corollary 3.10. *If* $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{L})$*, then* $\mathrm{LIST}(\mathrm{NL}, \mathrm{TAUT}, \mathrm{NL})$*.*

Proof: If $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{L})$, then $\mathrm{DTC}_{\mathrm{inv}}$ is an L-bounded logic for L and hence by part (c) of the previous corollary, $\mathrm{TC}_{\mathrm{inv}}$ is an NL-bounded logic for NL. Therefore, $\mathrm{LIST}(\mathrm{NL}, \mathrm{TAUT}, \mathrm{NL})$ by Theorem 3.8(b). □

Proof of Theorem 3.8: (a) Assume that $\mathrm{LIST}(\mathrm{L}, \mathrm{TAUT}, \mathrm{C})$. We show that $\mathrm{ACC}_{\leq} \in \text{co-}\mathrm{XC}_{\mathrm{uni}}$. Then $L(\mathrm{C})$-INV $\in \mathrm{XC}_{\mathrm{uni}}$ by Proposition 3.6; thus Lemma 3.7 yields our claim.

The problem $\mathrm{ACC}_{\leq}$ is in NP. Hence there is a logspace reduction $(\mathbb{M}, n) \mapsto \alpha(\mathbb{M}, n)$ from $\mathrm{ACC}_{\leq}$ to SAT; in particular, we have

$$(\mathbb{M}, n) \in \mathrm{ACC}_{\leq} \iff \alpha(\mathbb{M}, n) \in \mathrm{SAT},$$

or, equivalently,

$$(\mathbb{M}, n) \notin \mathrm{ACC}_{\leq} \iff \neg\alpha(\mathbb{M}, n) \in \mathrm{TAUT}. \tag{7}$$

Moreover we may assume that from $\alpha(\mathbb{M}, n)$ we can recover $(\mathbb{M}, n)$ in logarithmic space. It follows that:

Claim 1. Let $\mathbb{M}$ be a nondeterministic Turing machine that does not accept the empty input tape. Then

$$\{\neg\alpha(\mathbb{M}, n) \mid n \geq 1\}$$

is an L-subset of TAUT. ⊣

Let $\mathbb{S}$ be the algorithm that on input $\mathbb{M}$ by systematically going through all runs of length 1, all of length 2,... computes

$$n(\mathbb{M}) := \text{the least } n \text{ such that } \mathbb{M} \text{ accepts the empty input in } n \text{ steps.}$$

If $\mathbb{M}$ does not accept the empty input tape at all, then $\mathbb{S}$ does not stop and $n(\mathbb{M})$ is not defined.

By the assumption LIST(L, TAUT, C) there is a listing $\mathbb{L}$ of the L-subsets of TAUT by C-machines. We give the proof for the case that C is a space class (that is, C = L or C = NL). The proof for P is similar. Using (7) it is easy to verify that the following algorithm $\mathbb{A}$ accepts co-$\text{ACC}_{\leq}$.

$\mathbb{A}(\mathbb{M}, n)$

1. $d \leftarrow 1$;
2. In parallel simulate $\mathbb{S}$ with input $\mathbb{M}$ and the listing algorithm $\mathbb{L}$ using at most space $d \cdot \log n$;
3. **if** $\mathbb{S}$ stops with output $n(\mathbb{M})$ **then**
4. **if** $n(\mathbb{M}) > n$ **then** accept **else** reject;
5. **if** $\mathbb{L}$ lists a machine $\mathbb{M}'$ **then** simulate $\mathbb{M}'$ on input $\neg\alpha(\mathbb{M}, n)$ using at most space $d \cdot \log n$
6. **if** $\mathbb{M}'$ accepts **then** accept;
7. **if** $\mathbb{S}$ or $\mathbb{L}$ needs space $> d \cdot \log n$ **then** $d \leftarrow d + 1$; goto 2.

We still have to show that for fixed $\mathbb{M}$ the algorithm $\mathbb{A}$, on input $(\mathbb{M}, n) \notin \text{ACC}_{\leq}$, accepts in space $O(\log |n|)$.

Case "$\mathbb{M}$ accepts the empty input tape:" Then $\mathbb{S}$ will stop eventually, the space it uses only depends on its input $\mathbb{M}$. Hence, $\mathbb{A}$ only needs space $O(\log |n|)$.

Case "$\mathbb{M}$ does not accept the empty input tape:" By Claim 1, the set $\{\neg\alpha(\mathbb{M}, m) \mid m \geq 1\}$ is a L-subset of TAUT. Therefore, a machine $\mathbb{M}'$ accepting this set in logarithmic space will eventually be listed by $\mathbb{L}$. Then, one easily sees that $\mathbb{A}$ accepts such inputs in space $O(\log |n|)$.

(b) Now we turn to the proof of the converse (this proof is a generalization of the proof of Lemma 7 in [5]). We assume that $\text{C} \subseteq \text{C}'$ and that $L(\text{C})_{\text{inv}}$ is a C′-bounded logic for C. It is easy to introduce a vocabulary τ such that in logarithmic space we can associate with every propositional formula α a τ-structure $\mathcal{A}(\alpha)$ such that

(i) every propositional variable X of α corresponds to two distinct elements a_X, b_X of $\mathcal{A}(\alpha)$ and there is a unary relation symbol $P \in \tau$ such that $P^{\mathcal{A}(\alpha)} = \{a_X \mid X \text{ variable of } \alpha\}$;

(ii) the class

$$\{\mathcal{B} \mid \mathcal{B} \cong \mathcal{A}(\alpha) \text{ for some } \alpha \in \text{PROP}\}$$

of τ-structures is axiomatizable by a DTC[τ]-sentence and therefore by an $L(\text{C})[\tau]$-sentence $\varphi(\text{PROP})$;

(iii) if $\mathcal{B} \models \varphi(\text{PROP})$, then one can determine the unique $\alpha \in \text{PROP}$ with $\mathcal{B} \cong \mathcal{A}(\alpha)$ in logarithmic space.

Note that an ordered $\tau_<$-structure of the form $(\mathcal{A}(\alpha), <)$ yields an assignment of the variables of α, namely the assignment sending a variable X to TRUE if and only if $a_X < b_X$. As in logarithmic space we can check whether this assignment satisfies α there is a DTC[$\tau_<$]-sentence and hence an $L(\text{C})[\tau_<]$-sentence

$\varphi(\text{sat})$ that for every $\alpha \in \text{PROP}$ expresses in $(\mathcal{A}(\alpha), <)$ that the assignment given by $<$ satisfies α. We introduce the $L(\text{C})[\tau_<]$-sentence

$$\varphi_0 := \big(\varphi(\text{PROP}) \to \varphi(\text{sat})\big).$$

Then φ_0 is an $L(\text{C})_{\text{inv}}[\tau]$-sentence. Every assignment of α can be obtained by some ordering $<$ of $A(\alpha)$. Hence, by the definition of $\models_{L(\text{C})_{\text{inv}}}$, we see that for every $\alpha \in \text{PROP}$ and every $L(\text{C})_{\text{inv}}[\tau]$-sentence φ

$$\text{if } \mathcal{A}(\alpha) \models_{L(\text{C})_{\text{inv}}} (\varphi_0 \wedge \varphi), \text{ then } \alpha \in \text{TAUT}. \tag{8}$$

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

$$Q(\varphi) := \big\{\alpha \in \text{PROP} \mid \mathcal{A}(\alpha) \models_{L(\text{C})_{\text{inv}}} (\varphi_0 \wedge \varphi)\big\}.$$

We claim that the class of sets $Q(\varphi)$, where φ ranges over all $L(\text{C})_{\text{inv}}$-sentences coincides with the C-subsets of TAUT.

First let Q be a C-subset of TAUT. If Q is finite, it is easy to see that $Q = Q(\varphi)$ for some $\varphi \in L(\text{C})_{\text{inv}}$. Now let Q be infinite. The class

$$\big\{\mathcal{B} \mid \mathcal{B} \cong \mathcal{A}(\alpha) \text{ for some } \alpha \in Q\big\}$$

is in C (by (ii) and (iii) as L $\subseteq$ C), and therefore it is axiomatizable by an $L(\text{C})_{\text{inv}}[\tau]$-sentence φ. As the class contains arbitrarily large structures, the formula φ is invariant. We show that $Q = Q(\varphi)$.

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

Conversely fix $\varphi \in L(\text{C})_{\text{inv}}[\tau]$. By (8), we have $Q(\varphi) \subseteq \text{TAUT}$. By assumption, $L(\text{C})_{\text{inv}}$ is a C$'$-bounded logic for C. In particular, $L(\text{C})_{\text{inv}}$ is a logic for C and hence we have $Q(\varphi) \in \text{C}$. As $L(\text{C})_{\text{inv}}$ is C$'$-bounded, the algorithm $\mathbb{A}$ for the satisfaction relation (cf. Definition 3.1(b)) restricted to $\varphi_0 \wedge \varphi$ yields an algorithm of type C$'$ accepting $Q(\varphi)$.

Hence, the classes $Q(\varphi)$ where φ ranges over all $L(\text{C})_{\text{inv}}[\tau]$-sentences yield the listing witnessing $\text{LIST}(\text{C}, \text{TAUT}, \text{C}')$. □

The order-invariant first-order logic FO_{inv}. By the methods used in [4] one can show that DTC_{inv} already is a L-bounded logic for L if there is an algorithm deciding $\models_{\text{FO}_{\text{inv}}}$, which for fixed first-order sentence φ requires logarithmic space. We shall prove this result in the final version of this paper.

4. Listings, optimal proof systems, and almost optimal algorithms

In [16] Sadowski shows that the existence of listings of the P-subsets of TAUT by P-machines is equivalent to the existence of p-optimal proof systems for TAUT, and hence (by [13]) equivalent to the existence of almost optimal algorithms for TAUT. An extension of this result to other Q (instead of TAUT) are derived in [2]. Here we prove the corresponding results for L instead of P. We already state the result, even though we haven't introduced all concepts appearing in it so far.

Theorem 4.1. *For $Q \subseteq \Sigma^*$ with padding the following are equivalent:*

(a) Q has a space optimal logspace proof system.

(b) Q has an almost space optimal algorithm.

(c) $\text{LIST}(\text{L}, Q, \text{L})$.

For the reader familiar with the results of the previous section, we state a consequence of this result and of Corollary 3.9(a).

Corollary 4.2. $\mathrm{DTC}_{\mathrm{inv}}$ *is an* L*-bounded logic for* L *if and only if there is a space optimal logspace propositional proof system (that is a space optimal logspace proof system for* TAUT*).*

Corollary 4.3. *Let $Q \subseteq \Sigma^*$ with padding. If Q has an almost space optimal algorithm, then it has an almost optimal algorithm.*

Proof: One can generalize the proof of the result mentioned at the beginning of this section and show that

$$Q \text{ has an almost optimal algorithm } \iff \textsc{List}(\mathrm{P}, Q, \mathrm{P}).$$

As $\textsc{List}(\mathrm{L}, Q, \mathrm{L})$ implies $\textsc{List}(\mathrm{P}, Q, \mathrm{P})$ by Proposition 2.3 (a), our claim follows by Theorem 4.1. □

Let $\mathbb{A}$ be a deterministic algorithm. For every $x \in \Sigma^*$ let $t_{\mathbb{A}}(x)$ be the number of steps of the run of $\mathbb{A}$ on input x. Similarly, we denote by $s_{\mathbb{M}}(x)$ the space required by $\mathbb{M}$ on x. If $\mathbb{M}$ on x does not stop, then $t_{\mathbb{M}}(x) = \infty$, even though $s_{\mathbb{M}}(x)$ may be finite.

For every $x \in \Sigma^*$ the number of configurations of $\mathbb{A}$ on input x is bounded by

$$|x| \cdot 2^{O(s_{\mathbb{A}}(x))}. \tag{9}$$

Hence, if $s_{\mathbb{A}}(x)) \geq O(\log |x|)$, we may assume that

$$t_{\mathbb{A}}(x) \leq |x| \cdot 2^{O(s_{\mathbb{A}}(x))} \tag{10}$$

and if $\mathbb{A}$ outputs a string $\mathbb{A}(x)$ on input x, that

$$|\mathbb{A}(x)| \leq |x| \cdot 2^{O(s_{\mathbb{A}}(x))}. \tag{11}$$

In fact, at every step of the run of $\mathbb{A}$ on an input x, we record the space s and the time t the algorithm $\mathbb{A}$ has used so far. For some appropriate constant $c \in \mathbb{N}$ if

$$t > |x| \cdot 2^{c \cdot s},$$

then the algorithm $\mathbb{A}$ must be in some configuration it has already been before. Hence $\mathbb{A}$ will run forever and we can stop it and let it reject. To store such s and t, we only need additional space

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

4.1. Almost space optimal algorithms and listings. In this part we prove a first implication contained in Theorem 4.1, namely the implication (b) ⇒ (c). An algorithm deciding Q is almost space optimal if no other algorithm saves, compared with it, superlogarithmic space; more precisely:

Definition 4.4. A deterministic algorithm $\mathbb{A}$ deciding Q is *almost space optimal* for Q if for every deterministic algorithm $\mathbb{B}$ which decides Q there is a $d \in \mathbb{N}$ such that for all $x \in Q$

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

(note that nothing is required of the relationship between $s_{\mathbb{A}}(x)$ and $s_{\mathbb{B}}(x)$ for $x \notin Q$).

Proposition 4.5. *Let $Q \subseteq \Sigma^*$. If there is an almost space optimal algorithm for Q, then* $\textsc{List}(\mathrm{L}, Q, \mathrm{L})$.

Proof: Let $\mathbb{A}$ be an almost space optimal algorithm for Q. For every $d \in \mathbb{N}$, let $\mathbb{A}(d)$ be the algorithm $\mathbb{A}$ restricted to space $d \cdot \log n$, that is, $\mathbb{A}(d)$ on input x simulates $\mathbb{A}$ on input x but rejects if the simulation has to exceed space $d \cdot \log |x|$. Clearly,

(a) $L(\mathbb{A}(d))$ is an L-subset of Q for every $d \in \mathbb{N}$;

moreover, we show

(b) for every L-subset X of Q there is a $d \in \mathbb{N}$ with $X \subseteq L(\mathbb{A}(d))$.

In fact, let $\mathbb{B}$ be a deterministic algorithm deciding X and requiring space $O(\log\, n)$. Then the following algorithm $\mathbb{C}$ decides Q: On input x, in parallel it simulates $\mathbb{A}$ and $\mathbb{B}$ on input x; if $\mathbb{B}$ stops first and accepts, then $\mathbb{C}$ accepts, otherwise it answers as $\mathbb{A}$. Note that there is $c \in \mathbb{N}$ such that $s_{\mathbb{C}}(x) \leq c \cdot \log\, |x|$ for $x \in X$.

Then, by the almost space optimality of $\mathbb{A}$ there is a $d \in \mathbb{N}$ with

$$s_{\mathbb{A}}(x) \leq d \cdot (s_{\mathbb{C}}(x) + \log\, |x|)$$

for all $x \in Q$ and thus for all $x \in X$

$$s_{\mathbb{A}}(x) \leq d \cdot (c+1) \cdot \log\, |x|.$$

Therefore, $X \subseteq L(\mathbb{A}(d \cdot (c+1)))$.

We fix an effective enumeration $\mathbb{D}_1, \mathbb{D}_2, \ldots$ of all logspace deterministic Turing machines. Then, by (a) and (b), $(\mathbb{D}_i(\mathbb{A}(j))_{i,j\geq 1}$ is a listing of the L-subsets of L, where $\mathbb{D}_i(\mathbb{A}(j))$ on input x, first simulates $\mathbb{A}(j)$ and if this algorithm accepts, then it simulates $\mathbb{D}_i$ on input x and answers accordingly. □

4.2. Listings and space optimal logspace proof systems. We start by generalizing the notions of proof system and p-optimal proof systems to the logspace case and then prove implication (c) ⇒ (a) of Theorem 4.1.

Definition 4.6. (1) A *logspace proof system* for Q is a surjective function $P : \Sigma^* \to Q$ computable in logarithmic space.

We often tacitly identify a logspace proof system P with an algorithm witnessing its logspace computability.

(2) Let $P, P' : \Sigma^* \to Q$ be logspace proof systems for Q. We say that P *logspace simulates* or ℓ-*simulates* P' if there exists a logspace computable function $g : \Sigma^* \to \Sigma^*$ such that for every $w \in \Sigma^*$

$$P(g(w)) = P'(w).$$

(3) A logspace proof system for Q is *space optimal* if it ℓ-simulates every logspace proof system for Q.

Recall [6] that a proof system for Q is a surjective function $P : \Sigma^* \to Q$ *computable in polynomial time* and that a proof system P is p-optimal if for every proof system P' there is a function g as in (2) of the previous definition but now computable in polynomial time. Of course, every space optimal logspace proof system for Q is a p-optimal proof system for Q.

Most natural proof systems are logspace. Moreover, for every proof system P we get a logspace proof system P' defined by

$$P'(w) := \begin{cases} P(w_1), & \text{if } w = \langle w_1, c_1 \rangle, \text{ where } c_1 \text{ is the computation of } P \text{ on } w_1 \\ y_0, & \text{otherwise,} \end{cases}$$

where y_0 is a fixed element of Q. However, the transition from P to P' may destroy its space optimality. For the reader familiar with Frege systems we mention that any two Frege systems over the same set of propositional connectives ℓ-simulate each other.

We turn to the main result of this part.

Proposition 4.7. *Assume Q is nonempty and has a padding function. If* $\textsc{List}(\mathrm{L}, Q, \mathrm{L})$*, then Q has a space optimal logspace proof system.*

Proof: Fix $q_0 \in Q$ and let $\mathbb{L}$ be a listing of the L-subsets of Q by L-machines. We say that $v \in \Sigma^*$ is a *proof string* if it has the form

$$v = \langle \mathbb{D}, w, y, c, \mathbb{D}', c', c'' \rangle,$$

where

(S1) $\mathbb{D}$ is a deterministic Turing machine that on input w outputs y by the computation c;

(S2) The (partial) computation c' of $\mathbb{L}$ lists $\mathbb{D}'$;

(S3) $\mathbb{D}'$ accepts $\mathit{pad}(y, w)$ by the computation c''.

Clearly we can decide in logarithmic space whether a string $v \in \Sigma^*$ is a proof string. Moreover, if v is a proof string, then $y \in Q$ (as $L(\mathbb{D}') \subseteq Q$ and $\mathbb{D}'$ accepts $\mathit{pad}(y, w)$). Therefore the function P defined on Σ^* by

$$P(v) := \begin{cases} y, & \text{if } v \text{ is a proof string and } y \text{ is its third component} \\ q_0, & \text{otherwise} \end{cases}$$

is computable in logspace and has a subset of Q as range. So, P is a logspace proof system for Q, if we can show that every $y \in Q$ is in its range: As $\{\mathit{pad}(y, y)\}$ is an L-subset of Q, a machine $\mathbb{D}'$ accepting $\{\mathit{pad}(y, y)\}$ is listed by $\mathbb{L}$, say, by the computation c'. Then $P(v) = y$ for

$$v = \langle \mathbb{D}_{\mathrm{id}}, y, y, c, \mathbb{D}', c', c'' \rangle$$

where $\mathbb{D}_{\mathrm{id}}$ is a machine that on input w outputs w, the string c is its computation on input y, and c'' is the computation of $\mathbb{D}'$accepting $\mathit{pad}(y, y)$.

It remains to show that P is space optimal. For this purpose let $P' : \Sigma^* \to Q$ be a logspace proof system for Q. Then,

$$\mathit{Graph}(P') := \big\{ \mathit{pad}(y, w) \mid y, w \in \Sigma^* \text{ and } P'(w) = y \big\}$$

is an L-subset of Q. Hence, there is a computation c' of $\mathbb{L}$ listing a machine $\mathbb{D}'$ that decides $\mathit{Graph}(P)$. We define the function $g : \Sigma^* \to \Sigma^*$ by

$$g(w) := \langle P', w, P'(w), c, \mathbb{D}', c', c'' \rangle,$$

where c is the computation of P' on w and c'' is the computation of $\mathbb{D}'$ on $\mathit{pad}(y, w)$. It is easy to check that $g(w)$ is a proof string with

$$P(g(w)) = P'(w).$$

As g is computable in logspace, this finishes the proof. □

4.3. Space optimal proof systems and almost space optimal algorithms. In this part we show the following result, which together with Proposition 4.5 and Proposition 4.7 yields Theorem 4.1.

Proposition 4.8. *If Q has a space optimal logspace proof system, then it has an almost space optimal algorithm.*

For the proof of this proposition, we need the following result, which is the space-analogue of a theorem for P due to Levin [14]. We give a proof of this space-analogue in Section 5.

Definition 4.9. 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 $\mathbb{A}$ can behave arbitrarily.

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

Theorem 4.10. *Let $f : \Sigma^* \to \Sigma^*$ be a function that can be computed by an algorithm $\mathbb{F}$. 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 of Proposition 4.8: Let $P : \Sigma^* \to Q$ be a space optimal logspace proof system for Q and fix $y_0 \in Q$. We define a function $f : \Sigma^* \to \Sigma^*$ by

$$f(\langle \mathbb{A}, y, c, \mathbb{B}, c' \rangle) := y$$

if

(F1) $\mathbb{A}$ is a deterministic algorithm that accepts $y \in \Sigma^*$ by the computation c;

(F2) $\mathbb{B}$ is a deterministic algorithm that on input (y, c) computes a string w with $P(w) = y$; moreover, c' is the computation of $\mathbb{B}$ on input (y, c).

Otherwise, we set $f(w) := y_0$. It is easy to verify that the range of f is Q, in particular, (F2) guarantees that it is a subset of Q. Moreover there is an algorithm $\mathbb{F}$ that computes the function f in logarithmic space.

By Theorem 4.10, we have an optimal space inverter $\mathbb{O}$ for f such that for every algorithm $\mathbb{I}$ which inverts f and for every $y \in Q$ we have

$$s_{\mathbb{O}}(y) \leq O\big(\log |y| + s_{\mathbb{I}}(y) + \log |\mathbb{I}(y)| + s_{\mathbb{F}}(\mathbb{I}(y))\big) \leq O\big(\log |y| + s_{\mathbb{I}}(y) + \log (|\mathbb{I}(y)|)\big), \tag{12}$$

where the second inequality holds as $\mathbb{F}$ is a logspace algorithm; for $y \notin Q$ we have $s_{\mathbb{O}}(y) = \infty$.

Let $\mathbb{Q}$ be any algorithm deciding Q. We claim that the following algorithm $\mathbb{S}$ is an almost space optimal algorithm deciding Q.

$\mathbb{S}(x)$

1. $\ell \leftarrow 1$
2. simulate $\mathbb{Q}$ and the optimal inverter $\mathbb{O}$ on y in parallel using space at most ℓ
3. **if** both simulations exceed space ℓ, **then** $\ell \leftarrow \ell + 1$ and goto 2
4. **if** $\mathbb{Q}$ halts, **then** output accordingly
5. **if** $\mathbb{O}$ halts, **then** accept.

Clearly, $\mathbb{S}$ decides Q and for $y \in Q$ we have

$$s_{\mathbb{S}}(y) \leq O(s_{\mathbb{O}}(y)). \tag{13}$$

We claim that $\mathbb{S}$ is almost space optimal. For this purpose let $\mathbb{A}$ be any algorithm that decides Q. We get a logspace proof system $P_{\mathbb{A}}$ for Q by setting

$$P_{\mathbb{A}}(w) := \begin{cases} y, & \text{if } w = \langle y, c \rangle \text{ and the algorithm } \mathbb{A} \text{ accepts } y \text{ by the computation } c; \\ y_0, & \text{otherwise.} \end{cases}$$

Since P is a space optimal logspace proof system for Q, there is a logspace algorithm $\mathbb{B}$ such that for every $y \in Q$ accepted by $\mathbb{A}$ with computation c,

$$\mathbb{B} \text{ on input } \langle y, c \rangle \text{ computes a string } w \in \Sigma^* \text{ such that } P(w) = P_{\mathbb{A}}(\langle y, c \rangle) = y.$$

Using $\mathbb{A}$ and $\mathbb{B}$ we define the following inverter $\mathbb{I}$ of the function f:

$\mathbb{I}(y)$

1. Simulate the algorithm $\mathbb{A}$ on y
2. **if** $\mathbb{A}$ rejects y, **then** do not halt
3. **if** $\mathbb{A}$ accepts y with computation c
4. **then** output $\langle \mathbb{A}, y, c, \mathbb{B}, c' \rangle$, where c' is the computation of $\mathbb{B}$ on $\langle y, c \rangle$.

Let $y \in Q$. Then, as $\mathbb{B}$ is logspace computable, we have

$$\begin{aligned} s_{\mathbb{I}}(y) &\leq O(s_{\mathbb{A}}(y)) + O(\log |\langle y, c\rangle|) \\ &= O(s_{\mathbb{A}}(y)) + O\big(\log\, (|y| \cdot 2^{O(s_{\mathbb{A}}(y))})\big) \qquad \text{(by (9))} \\ &= O(s_{\mathbb{A}}(y)) + O(\log |y|), \end{aligned}$$

and hence by (11)

$$|\mathbb{I}(y)| \leq |y|^{O(1)} \cdot 2^{O(s_{\mathbb{A}}(y))}.$$

Moreover by the optimality of the inverter $\mathbb{O}$, i.e., by (12),

$$s_{\mathbb{O}}(y) \leq O\big(\log |y| + s_{\mathbb{I}}(y) + \log\, (|\mathbb{I}(y)|)\big) = O(s_{\mathbb{A}}(y) + \log |y|).$$

Finally, by (13), we get the inequality witnessing the almost space optimality of $\mathbb{S}$

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

4.4. Some variants. In Section 2 we introduced the concept of $\textsc{List}(\mathrm{C}, Q, \mathrm{C}')$ for various choices of C and C′. So far, in this section we only have considered the case $\mathrm{C} = \mathrm{C}' = \mathrm{L}$. For some other choices, Theorem 4.1 survives if the corresponding concepts of optimality are defined in the appropriate way. Here we only mention one result.

A *nondeterministic* algorithm $\mathbb{A}$ accepting Q is *almost space optimal* for Q if for every nondeterministic algorithm $\mathbb{B}$ which decides Q there is a $d \in \mathbb{N}$ such that for all $x \in Q$

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

Theorem 4.11. *Assume that Q has padding. Then*

$$\textsc{List}(\mathrm{NL}, Q, \mathrm{NL}) \iff \textit{Q has a nondeterministic almost space optimal algorithm.}$$

Proof: The direction from right to left is obtained along the lines of the proof of Proposition 4.5. Now 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.

$\mathbb{O}(x)$
// $x \in \Sigma^*$ an instance for Q

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

We show that $\mathbb{O}$ is nondeterministic almost space optimal. To that end, let $\mathbb{A}$ be a nondeterministic algorithm accepting Q. We consider the following subset of Q.

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

Using 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_0th machine $\mathbb{M}_{i_0}$ listed by $\mathbb{L}$ accepts $\mathit{LOG}(\mathbb{A})$ in space $O(\log n)$; in particular,

$$s_{\mathbb{M}_{i_0}}(\mathit{pad}(x, x01^d)) \leq O\big(\log |\mathit{pad}(x, x01^d)|\big) = O\big(\log\, (|x|^{O(1)} + d^{O(1)})\big) = O\big(\log |x| + \log d\big). \quad (14)$$

The first 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_0 (in Line 1) and $2^{s_{\mathbb{A}}(x)}$(in Line 3). This choices show that

$$s_{\mathbb{O}}(x) \leq O\left(c + s_{\mathbb{A}}(x) + \log |x| + \log \left|\mathit{pad}(x, x01^d)\right| + s_{\mathbb{M}_{i_0}}(\mathit{pad}(x, x01^d))\right), \quad (15)$$

where c counts the space for guessing i_0 and for computing the machine $\mathbb{M}_{i_0}$. By (14) and (15) we conclude that

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

Corollary 4.12. *Let $Q \subseteq \Sigma^*$ with padding. If Q has a nondeterministic almost space optimal algorithm, then it has a nondeterministic almost optimal algorithm.*

Proof: One can generalize the proof in [16] of a result for TAUT and show that

$$Q \text{ has a nondeterministic almost optimal algorithm } \iff \text{LIST}(\text{NP}, Q, \text{NP}).$$

Now the claim follows from the previous theorem using Proposition 2.3 (b). □

5. Proof of the space version of Levin's Theorem.

So, in this section we prove Theorem 4.10. We start by introducing a notation.

Let $\mathbb{A}$ and $\mathbb{A}'$ be algorithms computing (partial) functions f and f' from $\Sigma^* \to \Sigma^*$. By $\mathbb{A}; \mathbb{A}'$ we denote an algorithm that computes the function $f' \circ f$, i.e., $x \mapsto f'(f(x))$. Using the well-known trick of complexity theory, where one does not store the full intermediate string $f(x)$, one can choose $\mathbb{A}; \mathbb{A}'$ in such a way that

$$s_{\mathbb{A};\mathbb{A}'}(x) = O\big(s_{\mathbb{A}}(x) + \log |\mathbb{A}(x)| + s_{\mathbb{A}'}(\mathbb{A}(x))\big) \tag{16}$$

Proof of Theorem 4.10: Let $f : \Sigma^* \to \Sigma^*$ be a function that can be computed by an algorithm $\mathbb{F}$. Let $\mathbb{O}$ be the following program

```
𝕆(x)
1. k ← 0
2. k ← k + 1
3. for every program 𝕎 ∈ Σ* with |𝕎; 𝔽| ≤ k do
4.         simulate 𝕎; 𝔽 on input x using at most k − |𝕎; 𝔽| space
5.                 if the simulation outputs x
6.                   i.e., 𝕎 computes a string w with f(w) = x
7.                         then simulate 𝕎 on x (outputting the w) and halt
8. goto 2.
```

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) in Theorem 4.10. 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 the algorithm $\mathbb{I}; \mathbb{F}$ on input x is

$$\leq c + \log |\mathbb{I}; \mathbb{F}| + \log |x| + s_{\mathbb{I};\mathbb{F}}(x). \tag{17}$$

Clearly, we get an upper bound on the space that $\mathbb{O}$ requires on input x if we assume that k in Line 2 of $\mathbb{O}$ gets a value such that $|\mathbb{I}; \mathbb{F}| \leq k$ and the simulation of $\mathbb{I}; \mathbb{F}$ on input x can be performed with space at most $k - |\mathbb{W}; \mathbb{F}|$. Hence, we have

$$\begin{aligned}
s_{\mathbb{O}}(x) &\leq |\mathbb{I}; \mathbb{F}| + c + \log |\mathbb{I}; \mathbb{F}| + \log |x| + s_{\mathbb{I};\mathbb{F}}(x) && \text{(by (17))} \\
&\leq 2 \cdot |\mathbb{I}; \mathbb{F}| + c + \log |x| + O\big(s_{\mathbb{I}}(x) + \log |\mathbb{I}(x)| + s_{\mathbb{F}}(\mathbb{I}(x))\big) && \text{(by (16))} \\
&= O\big(\log |x| + s_{\mathbb{I}}(x) + \log |\mathbb{I}(x)| + s_{\mathbb{F}}(\mathbb{I}(x))\big). && \square
\end{aligned}$$

References

[1] D. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer Systems and Science* 41(3): 274-306, 1990.

[2] O. Beyersdorff and Z. Sadowski. Characterizing the existence of optimal proof systems and complete sets for promise classes. In *Proceedings of the 4th Computer Science Symposium in Russia (CSR'09)*, Lecture Notes in Computer Science 5675, pages 47–58, 2009.

[3] Y. Chen and J. Flum. A logic for PTIME and a parameterized halting problem. In *Proceedings of the 24th IEEE Symposium on Logic in Computer Science (LICS'09)*, pages 397–406, 2009.

[4] Y. Chen and J. Flum. On slicewise monotone parameterized problems and optimal proof systems for TAUT. In *Proceedings of Computer Science Logic (CSL'10)*, Lecture Notes in Computer Science 6247, pages 200–214, 2010.

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

[6] S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44:36–50, 1979.

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

[8] J. Hartmanis and L. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science 58* , 129–142, 1988.

[9] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[10] N. Immerman. Languages that capture complexit classes. *SIAM Journal on Computing*, 16: 770–778, 1987.

[11] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17: 935–938, 1988.

[12] W. Kowakczyk. Some connections between presentability of complexity classes and the power of formal systems of reasonning. In *Proceedings of Mathematical Foundations of Computer Science, (MFCS'88)*, pages 364–369, 1988.

[13] J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54:1063–1088, 1989.

[14] L. Levin. Universal search problems. *Problems of Information Transmission*, 9(3):265-266, 1973. In Russian; English translation in: B.A.Trakhtenbrot. A survey of Russian approaches to perebor (brute-force search) algorithms. *Annals of the History of Computing*, 6(4):384-400, 1984.

[15] A. Nash, J. Remmel, and V. Vianu. PTIME queries revisited. In *Proceedings of the 10th International Conference on Database Theory (ICDT'05)*, T. Eiter and L. Libkin (eds.), Lecture Notes in Computer Science 3363, 274–288, 2005.

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

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

ECCC ISSN 1433-8092

http://eccc.hpi-web.de