Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Computation 190 (2004) 196–219
Information and Computation
www.elsevier.com/locate/ic

The most nonelementary theory

Sergei Vorobyov*

Information Technology Department, Box 337, Uppsala University, 751 05 Uppsala, Sweden

Received 27 November 2002; revised 14 November 2003

Abstract

We give a direct proof by generic reduction that testing validity of formulas in a decidable rudimentary theory Ω of finite typed sets (Henkin, Fundamenta Mathematicæ 52 (1963) 323–344) requires space and time exceeding infinitely often

$$\exp_\infty(\exp(cn)) = 2^{\left.2^{\cdot^{\cdot^{2}}}\right\}height \quad 2^{cn}} \quad \text{for some constant } c > 0, \tag{1}$$

where n denotes the length of input. This gives *the highest* currently known lower bound for a decidable logical theory and affirmatively settles Problem 10.13 from (Compton and Henson, Ann. Pure Appl. Logic 48 (1990) 1–79):

> "Is there a "natural" decidable theory with a lower bound of the form $\exp_\infty(f(n))$, where f is not linearly bounded?"

The highest previously known lower (and upper) bounds for "natural" decidable theories, like *WS1S*, *S2S*, are of the form $\exp_\infty(dn)$, with *just linearly growing* stacks of twos. Originally, the lower bound (1) for Ω was settled in (12th Annual IEEE Symposium on Logic in Computer Science (LICS'97), 1997, 294–305) using the powerful uniform lower bounds method due to Compton and Henson, and probably would never be discovered otherwise. Although very concise, the original proof has certain gaps, because the method was pushed out of the limits it was originally designed and intended for, and some hidden assumptions were violated. This results in slightly weaker bounds—the stack of twos in (1) grows subexponentially, but superpolynomially, namely, as $2^{c\sqrt{n}}$ for formulas with fixed quantifier prefix, or as $2^{cn/\log(n)}$ for formulas with varying prefix. The independent *direct* proof presented in this paper closes the gaps and settles the originally claimed lower bound (1) for the minimally typed, succinct version of Ω.
© 2004 Elsevier Inc. All rights reserved.

*Fax: +46-18-55-02-25.
E-mail address: Sergei.Vorobyov@csd.uu.se.
URL: http://user.it.uu.se/~vorobyov/.

0890-5401/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2004.02.002

AMS classification: MSC 68Q25 Analysis of algorithms and problem complexity; 03D15 Complexity of computation

Keywords: Lower complexity bound; Nonelementary theory; Generic reduction; Reduction via length order; Inductive definition

1. Introduction

Some nonelementary theories[1] are more nonelementary than others. Indeed, a theory with lower and upper time bounds of the form[2]

$$2^{\cdot^{\cdot^{2^n}}}\Big\}\log(\log(\log(\log(\log(\log n)))))$$

is, of course, nonelementary, but this is immaterial, because for all inputs one can ever encounter or even imagine in practice the function above is linear.

Other theories, like the well-known *weak monadic second-order theory of one successor WS1S*[3] or *two successors S2S* have lower and upper bounds of the form $\exp_\infty(dn)$, with linearly growing stacks of twos[4]; see [1,2,13,15] for surveys of known results.

The theory we consider in this paper is far more nonelementary. Type theory Ω is a rudimentary fragment of the theory of propositional types due to Henkin [3]. This is the higher-order theory of the stratified cumulative hierarchy over $\{0,1\}$ in the language $L = \{\in\}$ [8,11]; see Definition 1.

In this paper we directly prove by generic reduction the following.

Main theorem. *Any Turing machine deciding Ω requires space (hence time) exceeding*

$$\exp_\infty(\exp(d\cdot|S|)) = 2^{\cdot^{\cdot^{2}}}\Big\}\textit{height } \exp(d\cdot|S|)=2^{d\cdot|S|} \tag{2}$$

for some constant $d > 0$ and infinitely many sentences S of Ω.

Theorems 8, 10 below refine the Main Theorem for two different versions of Ω and for fixed quantifier prefixes. The lower bound (2) remains the same (with a different constant), no matter which reasonable computational model is used.[5]

---

[1] A theory (problem) is called *elementary in the sense of Kalmar* if it can be decided within time bounded above by a k-story (for k fixed) exponential function $\exp_k(n)$, where n is the length of input. The functions $\exp_m(n)$ are defined by $\exp_0(n) = n$ and $\exp_{m+1}(n) = 2^{\exp_m(n)}$. The usual exponential function $\exp(n) = 2^n$ coincides with $\exp_1(n)$, and the iterated exponential $\exp_\infty(n) = \exp_n(1)$.

[2] According to common practice, $\log(n)$ is a shorthand for $\max\{1, \lceil\log_2(n)\rceil\}$.

[3] The first one proved nonelementary by A. Meyer [10] in May 1972.

[4] A. Meyer [10] proved a weaker lower bound, with a logarithmically growing stack.

[5] All "reasonable" computational formalisms can be modeled by a Turing machine with only a polynomial slow-down.

One can wonder what is so interesting about the theory Ω and why it could be considered "natural." The author of this paper is not the first one who addressed the complexity of Ω. For example, A. Meyer [9, Theorem pp. 478–479, No. 7] claimed the

$$\left.\begin{matrix} 2^{\cdot^{\cdot^{2^n}}} \\ 2 \end{matrix}\right\}\varepsilon\cdot\log(n)\ \textit{height}$$

lower bound. Statman [11] claimed that Ω is nonelementary (without any explicit lower bounds) and used this fact to prove that β-equality in the simply typed lambda calculus is not elementary recursive. Later Mairson [8] sketched the proof that Ω is nonelementary, also without any explicit lower bounds. Note that Mairson's proof *does not imply* the lower bound (2). The high complexity of Ω came unnoticed until in [17] we settled, by using the method of Compton and Henson [1], the lower bound (2), and used it together with Statman's reduction to prove the tight $\exp_\infty(cn)$ lower bound for β-equality in the simply typed lambda calculus. This lower bound now *precisely* matches (with a different constant) the known upper bound of the form $\exp_\infty(dn)$ due to Tait.

As another important application, [17] shows that a long-standing, currently still open[6] *higher-order matching problem* in the simply typed lambda calculus due to Huet has a lower bound of the form $\exp_\infty(cn/\log(n))$. This provides an example asked for in [1, Problem 10.11]:

> "Give nontrivial lower bounds for mathematically interesting problems whose decidability is still open."

Vorobyov and Voronkov [18] used the lower bound (2) to show that determining whether a given nonrecursive logic program over sets *succeeds* has the same exponentially growing stack of twos $\exp_\infty(\exp(dn))$ as a lower bound.

Kuper and Vardi [6] and also Hull [5] considered similar formalisms of logical queries over sets with the *powerset* constructor. They proved tight lower and upper bounds of the form $\exp_\infty(cn)$, with *linearly growing* stacks of twos. The main reason of higher complexity of Ω is that its language is *exponentially more succinct*: it uses *binary notation* for types interpreted it terms of iterated powersets, whereas Kuper and Vardi [6] use *unary* notation for iterated powersets. For example, in their formalism saying that "x is an element of $\underbrace{\textit{powerset}\,(\ldots(\textit{powerset}}_{k\text{ times}}\,(\{0,1\})\ldots)$" requires $O(k)$ bits. The same thing in Ω is expressed by $x^{k-1} \in y^k \wedge \forall z^{k-1}(z^{k-1} \in y^k)$ and needs just $O(\log k)$ bits. It turns out that this exponential succinctness translates into the exponential speed-up in the growth of stacks of twos. Another reason is that in proving the lower bound for Ω we *(almost) do not need inductive definitions*, whereas they are used in [6] to define large sets. Sections 13, 17.4, and 19 discuss why inductive definitions in typed theories lead to *weaker* lower bounds.

Originally, the lower bound (2) in the Main Theorem was settled by using the powerful uniform lower bounds method of [1] in October 1996, and would probably never be discovered otherwise. Recall that it came unnoticed in [1,11,9,8]. Since the first report [17] on the lower bound (2), we felt necessary to provide an independent alternative proof in order to increase confidence in the validity of the claim, as well as of all applications we mentioned before, and dispel all suspicions

[6] **Added in proof.** Since this paper was submitted, R. Loader published the undecidability proof of the higher-order β-matching [Logic Journal of the IGPL, 11:1 (2003), pp. 51–68]. As of February 2004, the undecidability of $\beta\eta$-matching remains open. Our lower bound holds for both β- and $\beta\eta$-matching problems.

as to applicability of the method in the area it was not developed and intended for. This paper gives such an alternative proof by direct generic reduction, and also unveils a hidden assumption of Compton–Henson's method violated in [17].

Roughly, this "hidden" assumption is as follows. In first-order theories one can write formulas with *linearly* many quantifiers, but using only a *fixed* number of different variables, by reusing variable names. This allows for keeping the length of formulas *linear* in defining large ordered sets—the crucial property in proving strong lower bounds. This is *not necessarily true* for higher-order theories with variables keeping their *type annotations*. Indeed, while one can reuse variable names, the number of variable occurrences remains *linear*. If, additionally, variable types linearly depend on input, then one gets a *quadratic* blow-up in the length of formulas. This observation, applied uniformly to the method of [1], suggests that the lower bound for Ω proved in [17] should be lowered to a more modest $\exp_\infty(\exp(\sqrt{cn}))$ (note: still a superpolynomial stack of twos). However, as an additional advantage, the proof presented in this paper shows that Ω is capable of defining large ordered sets *without inductive definitions* that require linear number of variable occurrences leading to a quadratic explosion. This repairs a "slightly" incorrect application of Compton–Henson's method in [17], and saves the original claim.

Another advantage of the direct proof presented here is that it yields, as a by-product, an interesting result about a *fixed quantifier prefix complexity*. Usually one has to allow an arbitrary quantifier alternation depth in formulas to settle lower bounds. In Ω this can be done with a *fixed* quantifier prefix, with slightly weaker lower bounds; see Theorem 8. This came unnoticed in [17]. Results of this form cannot be obtained by using the uniform method of [1], and, to our knowledge Ω is the first example of a "natural" theory with this property.

Paper outline. After preliminaries, Section 3 presents the proof plan, and the sections that follow implement it. Section 16 makes an intermediate pause by presenting lower bounds for a fixed quantifier prefix, and the succeeding sections push up the lower bounds to the strongest possible. Some lower bound basics are moved to Appendix A.1.

2. Preliminaries

We assume the basic knowledge and notation concerning words, languages, complexity, reductions, asymptotics, etc. As usual, $\mathcal{P}(X)$ and $card(X)$ denote the set of all subsets of a set X and its cardinality, respectively, ω denotes the set of natural numbers. The function $\exp_\infty : \omega \to \omega$ is recursively defined by $\exp_\infty(0) = 1$ and $\exp_\infty(k+1) = 2^{\exp_\infty(k)}$. The m-story exponential functions $\exp_m(n)$ are defined by $\exp_0(n) = n$ and $\exp_{m+1}(n) = 2^{\exp_m(n)}$. Note that $\exp_\infty(n) = \exp_n(1)$. Throughout the paper we use $\exp_\infty(f(n))$ as a shorthand for $\exp_\infty(\lfloor f(n) \rfloor)$.

Type theory Ω is a very rudimentary fragment of the theory of propositional types due to Henkin [3], as defined by Statman [11] and Mairson [8].

Definition 1 (*Theory* Ω)**.** The language of *type theory* Ω is a language of set theory, where every variable has a natural number type, written as a binary superscript, and there are two constants $\mathbf{0}$, $\mathbf{1}$ of type 0. The atomic formulas of Ω are *stratified*, i.e., have form $\mathbf{0} \in x^1$, or $\mathbf{1} \in x^1$, or $x^n \in y^{n+1}$. All

other formulas are built by using $\neg$, $\wedge$, and $\forall$. The interpretation of Ω is as follows: **0** denotes 0, **1** denotes 1, and x^n ranges over $\mathcal{D}_n$, where $\mathcal{D}_0 = \{0, 1\}$ and $\mathcal{D}_{n+1} = \mathcal{P}(\mathcal{D}_n)$.

Note that $card(\mathcal{D}_i) = \exp_\infty(i+1)$. Decidability of Ω is immediate, because each quantifier runs over a *finite* domain. See Section 19 for the upper complexity bound.

Encoding. To argue about decision complexity, we fix an arbitrary reasonable encoding of formulas of Ω as binary strings and agree that a variable of Ω is represented by its type and its identification number within a type, both written in *binary*.

Verbose vs. succinct version of Ω. Annotating all variable occurrences in formulas of Ω with their types is redundant. For example, $x^k \in z^{k+1} \wedge y^k \in z^{k+1}$ can be unambiguously abbreviated to $x \in z^{k+1} \wedge y \in z$, because all missing type annotations in the last formula may be easily and uniquely reconstructed. Therefore, we distinguish between two versions of the theory Ω:

- *Fully typed, or verbose*, in which full type annotations are supplied for all variable occurrences.
- *Minimally (partially) typed, or succinct*, in which formulas are supplied with only a minimal type information allowing for an unambiguous reconstruction of the full type information about variables.

This distinction becomes important as soon as succinct reducibilities are concerned. Consider a conjunction $\bigwedge_{i=0}^{p} x_i^k \in Z^{k+1}$, where both p and the notational length of type k are $O(n)$. Then the length of the conjunction above is $O(n^2)$. The same conjunction written in succinct form $x_0 \in Z^{k+1} \wedge \bigwedge_{i=1}^{p} x_i \in Z$ has length $O(n \log n)$. As a consequence, succinct Ω has "slightly" (in fact, nonelementarily) higher lower bounds, as discussed below.

To avoid clutter, in writing formulas below we frequently and informally omit typing some variable occurrences, which may be easily and uniquely reconstructed.

Model of computation. We use the ordinary language recognizing deterministic Turing machine M with a semi-infinite (to the right) tape used both for input, work, and output. We assume without loss of generality that the tape alphabet Σ of M consists of two symbols, $\Sigma = \{0, 1\}$. We also apply all standard assumptions: that M always starts in its unique initial state observing the leftmost tape cell, that the input is always written on the left end of the tape, that M accepts by entering its unique accepting state q_a observing the leftmost cell after erasing all the tape space used in computation, etc.; see, e.g. [2,13]. The lower bounds we obtain routinely translate to other realistic models of computation, with only different constants. By $DSPACE(S(n))$ we denote the class of problems solvable by deterministic Turing machines in space $S(n)$.

Reducibilities. In order to settle the strongest lower bounds we need to use the tightest possible reductions. Assuming the reader has basic knowledge of lower bound techniques, here we only define *reducibility via length order* and state an important corresponding technical lemma used several times throughout the paper. To keep the paper self-contained, some details and proofs are moved to Appendix A.1.

Definition 2 (*Reducibility "via length order"*)**.** Say that a problem A is *polynomial time reducible to a problem B via length order* $g(n)$ if there exists a deterministic polynomial time computable function f and a constant $c > 0$ such that for all x in the language of A one has:

$$x \in A \Leftrightarrow f(x) \in B, \tag{3}$$

$$|f(x)| \leqslant c \cdot g(|x|) \text{ (except, maybe, finitely many } x). \tag{4}$$

Polynomial time reducibility via length order n is called *polynomial time linearly bounded reducibility*. □

Lemma 3. *Suppose, every problem in* $DSPACE(\exp_\infty(\exp(n) - 2))$ *is polynomial time reducible via length order* $g(n)$ *to a problem* T.
(1) *If* $g(n) = n$ *then* $T \notin DSPACE(\exp_\infty(\exp(dn)))$.
(2) *If* $g(n) = n\log(n)$ *then* $T \notin DSPACE(\exp_\infty(\exp(dn/\log(n))))$.
(3) *If* $g(n) = n^2$ *then* $T \notin DSPACE(\exp_\infty(\exp(d\sqrt{n})))$.
(*In each case* $d > 0$ *is some constant*.) □

Recall that $T \notin DSPACE(f(n))$ implies that every decision procedure for T requires space (hence, time) exceeding $f(n)$ on infinitely many inputs (lower bound for deciding T).

3. Proof plan

According to Lemma 3, our aim in the remainder of the paper is to show that every problem in $DSPACE(\exp_\infty(\exp(n)) - 2)$ is reducible to Ω, i.e., there exist a reduction f and a constant c satisfying (3) and (4) for the appropriate length order $g(n)$ depending on the version of Ω.

Let A be an arbitrary problem in $DSPACE(\exp_\infty(\exp(n)) - 2)$, and let M be a corresponding $(\exp_\infty(\exp(n)) - 2)$-space bounded TM deciding A. We will give a reduction f by constructing, for each x of length n in the language of A, the sentence $\Phi_{M,x}$ true in Ω iff M accepts x.

Remark 4. In constructing this reduction it is important that all parameters of A, represented by a TM M, are fixed before we start constructing f (these include the number of tape symbols, states, commands, etc.) and only influence the value of the constant c. This is crucial for the order of reduction. Otherwise, if the description of M were considered as a part of input, the number of triples of tape symbols may be *cubic* in the length of input; see Section 15. □

We start constructing the sentence $\Phi_{M,x}$ in Section 5, after extending the language of Ω by allowing explicit definitions.

We try to present enough technical details of the encoding. Although considered standard and well known to the small universe of people who did lower bound proofs, the reduction contains several subtle and nonstandard places, and applies in the typed context (which sometimes substantially differs from the untyped case [1,2]). We believe that keeping sufficient details is helpful to the reader, and yields a verifiable, easily reconstructible proof, understandable by a non-specialist. In any case, the proof presented below allowed the author to close several gaps in the preliminary report [17], which relied upon [1]. As a by-product, it also implies a new result on the superpolynomial-stack-of-twos lower bound for a fixed quantifier prefix class of Ω; see Theorem 8.

4. Using explicit definitions

Let us extend the language of Ω by allowing *explicit definitions*. This results in simpler and more intuitive formulas, but does not really increase the expressive power and complexity of the theory, because all explicit definitions can be eventually eliminated from any formula giving only a *linear* blow-up. This will not harm the linear boundedness of reductions we construct.

Set-theoretic notions. We need the usual set-theoretic explicit definitions like $x^m \subseteq y^m \equiv_{df} \forall z^{m-1}\ (z^{m-1} \in x^m \Rightarrow z^{m-1} \in y^m)$ for every $m \in \omega$, and similarly for strict subset $\subsetneq$, and set (in)equality.

Terms. The language of Ω does not have terms, except variables. Terms, like $\{x\}$, $\{x, y\}$, $\{\{x\}, \{x, y\}\}$, are useful representations for singletons, pairs, ordered pairs, which we will frequently need. Instead, we can define predicates for "to be a singleton, pair, ordered pair" by:

$$\begin{aligned}
\{x^n\} = y^{n+1} &\equiv_{df} x^n \in y^{n+1} \wedge \forall z^{n+1}(x^n \in z^{n+1} \Rightarrow y^{n+1} \subseteq z^{n+1}),\\
\{x^n, y^n\} = z^{n+1} &\equiv_{df} x^n \in z^{n+1} \wedge y^n \in z^{n+1} \wedge \forall w^{n+1}(x^n \in w^{n+1} \wedge y^n \in w^{n+1} \Rightarrow z^{n+1} \subseteq w^{n+1}),\\
\langle x^n, y^n\rangle = z^{n+2} &\equiv_{df} \exists u^{n+1} v^{n+1}\Big[\{x^n\} = u^{n+1} \wedge \{x^n, y^n\} = v^{n+1} \wedge \{u^{n+1}, v^{n+1}\} = z^{n+2}\Big]. \qquad (5)
\end{aligned}$$

For notational simplicity we will continue to use the term notation like $\{x^n\} = y^{n+1}$ instead of a less natural predicate notation *Is-Singleton*(y^{n+1}, x^n). Note how variables are typed in the explicit definitions above. Recall that by definition of Ω one *cannot* form a pair of elements of two different types. The explicit definitions above are not fully expanded (according to the usual mathematical practice), which can be done resulting only in a linear increase in the length of formulas.

Eliminating terms from formulas. We need to make the last explanation concerning the use of terms in formulas. Consider, for example, the formula (we omit types for simplicity), $\langle a, b\rangle \in \{c, d\}$, which translates into

$$\forall x, y(x = \langle a, b\rangle \wedge y = \{c, d\} \Rightarrow x \in y),$$

and two atoms in the premise should also be replaced by their explicit definitions. Such a transformation consists in introducing new variables corresponding to subterms, and putting their definitions of in the premise. Such a transformation can always be done routinely, giving only a *linear* increase in the length of formulas, *provided the depth of terms is bounded in advance*. Unbounded depth may result in quadratic blow-up, due to annotating linearly many variable occurrences (corresponding to linearly many subterms) with linearly long types.

Therefore, we can freely use all the above explicit definitions and terms in the constructions below, without running a risk to get more than a linear blow-up in the size of formulas.

5. Formula for an accepting computation

Given an arbitrary but fixed $(\exp_\infty(\exp(n)) - 2)$-space bounded TM M (cf., Remark 4) with tape alphabet $\Sigma = \{0, 1\}$, set of states $\mathcal{Q}$ ($\Sigma \cap \mathcal{Q} = \emptyset$), and an input $x \in \Sigma^+$ of length $n > 0$, we will construct the sentence

$$\Phi_{M,x} \equiv_{df} \exists R^{t+5} \left\{ \forall \mathcal{V} \Big[A \Rightarrow I(R) \wedge C(R) \wedge F(R) \wedge \forall R' \Big(I(R') \wedge C(R') \Rightarrow R \subseteq R' \Big) \Big] \right\}, \tag{6}$$

in the language of Ω, where:

(1) the variable R^{t+5} stands for a "run" of M, where t, called the *principal type* (to be defined in Section 10), *linearly depends* on the input length n; the existentially quantified occurrence R^{t+5} in (6) is the *only* variable occurrence needed to be annotated by a type—all other variables of (6) can be uniquely and unambiguously typed;
(2) A is an auxiliary formula discussed in Section 12, and $\forall \mathcal{V}$ quantifies over auxiliary variables;
(3) $I(R)$ says that R contains an initial *instantaneous description* (ID) of M on input x, defined in Sections 13 and 17;
(4) $C(R)$ says that R is closed with respect to transitions of the TM M, defined in Section 15;
(5) the $\forall R'$-quantified subformula in (6) expresses that R is a *minimal* set containing the initial ID and closed with respect to Ms transitions;
(6) $F(R)$ says that R contains an accepting ID of M, defined in Section 14;
(7) intuitively, the whole formula (6) says that there exists a path from the initial to the final configuration by using transitions of M, or, equivalently, that M accepts x.

6. Acceptance

By definition, an $(\exp_\infty(\exp(n)) - 2)$-space bounded TM M accepts an input x iff there exists a sequence of IDs, starting with an initial ID, with each succeeding ID obtained from the preceding one by applying one of the transition rules of M, and ending with an accepting ID. Since M is an $(\exp_\infty(\exp(n)) - 2)$-space bounded, we make a unifying assumption that all its IDs have equal length $\exp_\infty(\exp(n)) + 1$ and are of the form

$$\$d_1 \cdots d_{\exp_\infty(\exp(n))-1}\$, \tag{7}$$

where: (1) $\$ \notin \Sigma \cup \mathcal{Q}$ are tape end markers, over which M never tries to come across, (2) *exactly one* of d_is is a head state symbol (meaning that M is in the designated state observing the $i+1$st tape cell), and (3) the remaining $\exp_\infty(\exp(n)) - 2$ symbols are symbols of the M's tape alphabet and/or blanks; we assume that the tape unused by M is padded by blanks, and the blank symbol is not in $\Sigma \cup \mathcal{Q}$.

Thus the total (maximal) tape space described by (7) is $\exp_\infty(\exp(n)) - 2$.

7. Representing a run

We will represent a run R of a TM M as a set of pairs of IDs of M satisfying two properties:

(1) for all $\langle x, y\rangle \in R$ the ID y is obtained from the ID x in one step of M; (Elements of R are correct ID transitions of M.)
(2) if $\langle x, y\rangle \in R$ and y is not final, then for some z one has $\langle y, z\rangle \in R$. ($R$ is closed with respect to M transitions.)

Note that (6) stipulates that R is a minimal set satisfying these properties.

8. Representing an ID

An ID of an $(\exp_\infty(\exp(n)) - 2)$-space bounded TM M will be represented as a set of pairs $ID \subseteq L \times L$, where:

(1) L is an auxiliary linearly ordered set of cardinality $\exp_\infty(\exp(n)) + 1$ defined in Section 9, needed to index the symbols of an ID in (7),
(2) $\{x \mid \exists y\langle x, y\rangle \in ID\} = L$—to represent (7) we need a *total* function with the domain of cardinality $\exp_\infty(\exp(n)) + 1$,
(3) $card(\{y \mid \exists x\langle x, y\rangle \in ID\}) = card(\mathcal{Q} \cup \Sigma) + 2$—we need to represent states from $\mathcal{Q}$, tape symbols Σ, a blank, and the end marker \$ by elements of L.

Thus, an ID (7) is represented an L-indexed sequence of tape symbols (including a head state) represented as elements of L, padded by blanks to the length $\exp_\infty(\exp(n)) - 2$, and embraced by \$.

Recall that in Ω we can only construct sets of elements of the *same type*; see Section 4. That is why we use subsets of the Cartesian square of L to represent IDs. Note that an ID has type $t + 2$, if L is of type t; see (5). Similarly, an ordered pair of IDs has type $t + 4$. Consequently, a set R of such pairs has type $t + 5$. This explains typing in (6).

9. Large linearly ordered set

Define the predicates "to be linearly ordered" by

$$LO\ (X^t) \equiv_{df} \forall x, y(x \in X \wedge y \in X \Rightarrow (x \subseteq y \vee y \subseteq x)),$$

with type t defined in Section 10, and also "to be a maximal chain" by

$$MC\ (L^t) \equiv_{df} LO\ (L) \wedge \forall L'\Big(LO\ (L') \Rightarrow L' \subseteq L\Big). \tag{8}$$

Everywhere below L^t denotes a maximal chain, satisfying $MC(L^t)$ defined by (8). We need the following simple and useful.

Lemma 5. *Any maximal chain $S^t \in \mathcal{D}_t = \mathcal{P}(\mathcal{D}_{t-1})$ contains exactly $card(\mathcal{D}_{t-1}) + 1 = \exp_\infty(t) + 1$ elements.*

Proof. We may always suppose that the first and the last elements of any maximal chain $S^t \in \mathcal{D}_t$ are $\emptyset$ and $\mathcal{D}_{t-1}$. Otherwise, S can be extended by adding these elements and is non-maximal. Write the chain S as a sequence $X_0 \subset \cdots \subset X_i \subset X_{i+1} \subset \cdots \subset X_m$. We claim $m = card(\mathcal{D}_{t-1})$. Otherwise, one should have $card(X_{i+1} \backslash X_i) > 1$ for some i. Let $u \in X_{i+1} \backslash X_i$. Then the chain may be extended by adding $X_i \cup \{u\}$, i.e., $X_i \subset X_i \cup \{u\} \subset X_{i+1}$, and we get a contradiction. Clearly, S cannot have more

than $card(\mathcal{D}_{t-1})+1$ elements, because any pair $X_i \subset X_{i+1}$ of adjacent elements in S should have cardinalities differing at least by one. □

It is important to note that we succeeded to define a large linear order of size $\exp_\infty(t)+1$ *without any inductive definitions*, by a *fixed* formula of size $O(t)$, with only type t varying. This is one of the reasons Ω is so hard to decide.

Define the "successor in L^t" and the "three adjacent elements" predicates by:

$$\begin{array}{ll} succ(x,y,L^t) & \equiv_{df}\ x\in L\wedge y\in L\wedge x\neq y\wedge\forall z(x\subseteq z\subseteq y\Rightarrow(z\subseteq x\vee y\subseteq z)),\\ adj3(x,y,u,L^t) & \equiv_{df}\ succ(x,y,L)\wedge succ(y,u,L). \end{array} \tag{9}$$

10. The principal type

The existentially quantified variable R of (6) has type $t+5$, where t is the type (called *principal* in (6)) of the variable L^t denoting a maximal chain defined in Section 9. Section 8 explained why L^t should have cardinality $\exp_\infty(\exp(n))+1$, and from Lemma 5 we know that L^t has cardinality $\exp_\infty(t)+1$. Thus, the principal type t should be chosen as $t=1\underbrace{0\ldots0}_{n\text{ times}}$, which specifies L^t as a variable of type 2^n (recall that type annotations of variables in Ω are written in *binary*). This type annotation t for L^t defines uniquely the types of all other variables involved in $\Phi_{M,x}$, which will differ from t only by constants, with $t+5$ being the largest. This property will be provided by the construction of $\Phi_{M,x}$. Therefore, all variable type annotations in $\Phi_{M,x}$ will be *linearly bounded* in the length of input. Conversely, the largest type $t+5$ of the existentially quantified variable R of (6) uniquely defines the principal type t of L^t, as well as (smaller) types of all other variable occurrences in $\Phi_{M,x}$.

11. Tape, state, and auxiliary symbols

We need to use certain elements of the maximal chain L^t to represent tape, state, and auxiliary symbols, as explained in Section 8. It suffices to choose enough different fresh variables $v_1,\ldots,v_m$ of type $t-1$: one variable Q_i per state symbol $q_i\in\mathcal{Q}$, plus four variables *BLANK*, *END*, *ZERO*, *ONE*, for the blank, end marker, tape symbols $0,1\in\Sigma$, and to add $\bigwedge_{i=1}^m v_i\in L^t\ \wedge\bigwedge_{1\leqslant i\neq j\leqslant m} v_i\neq v_j$ to the auxiliary formula (10) we construct. We may assume without loss of generality that L is large enough to possess at least m elements; m is a constant, fixed when a TM for a problem is chosen; see Remark 4. The last formula above is almost a *fixed* formula depending only on a constant number of state and tape symbols in the description of M. However, each variable occurrence is assigned a type of length *linearly dependent* on the input length. Thus the above formula is of linear length.

This phenomenon repeats several times in the sequel and deserves a special.

Definition 6. Call a formula of Ω *quasi-fixed* if, after erasing all types of variables, it becomes a fixed formula, independent of input. □

Note that (8) defining L^t is quasi-fixed. We will construct, whenever possible, quasi-fixed formulas with variables annotated by types *linearly* depending on input; see Section 10. Thus, the sizes of such formulas will be *linear* in the length of input. If a formula of Ω is not quasi-fixed (e.g., contains a linear number of variable occurrences of non-fixed types), its size may grow *non-linearly* (e.g., quadratically) in the length of input. Therefore, since we need *linear bounded* reductions, we pay special care in constructing *quasi-fixed* formulas.

12. Auxiliary formula

We select fresh different variables $X_{fst}, X_{lst}, X_0, X_1$, and, as described above, fresh different variables *ZERO*, *ONE* (for the tape alphabet), *BLANK* (for the blank), *END* (for the end marker \$), and Q_i for all states $q_i \in \mathcal{Q}$. All these variables are of type $t-1$. The set $\mathcal{V}$ of all these variables is a finite set. Its size is a fixed constant determined by the problem. The auxiliary formula A in (6) is defined as

$$A \equiv_{df} MC\ (L^t) \wedge \min(X_{fst}, L) \wedge \max(X_{lst}, L) \wedge adj3(X_{fst}, X_0, X_1, L) \wedge \bigwedge_{V \in \mathcal{V}} V \in L \wedge \bigwedge_{\text{for different } V, V' \in \mathcal{V}} V \neq V', \tag{10}$$

where $\min(x, L)$ is explicitly defined by $x \in L \wedge \forall z(z \in L \Rightarrow x \subseteq z)$, and similarly for max. The formula (10) simply says that X_{fst}, X_{lst} are the first and the last elements in the chain L, X_0 is a successor to X_{fst}, X_1 is a successor to X_0, and all variables $V_i \in \mathcal{V}$ are interpreted as different elements of L.

13. Initial ID, subformula $I(R)$

Suppose that the TM M starts in the initial state q_0 observing the first symbols of the input sequence $s_1 \dots s_n \in \{0,1\}^+$. As a first approximation to represent the initial ID $\$q_0 s_1 \dots s_n b \dots b\$$ of M, let us select fresh different variables $X_2, \dots, X_n$, in addition to selected earlier, and write the following formula (with S_i equal *ZERO* when s_i is 0 and S_i equal *ONE* when s_i is 1):

$$\begin{aligned} IC(C^{t+2}) \equiv_{df}\ & \langle X_{fst}, END\rangle \in C \wedge \langle X_0, Q_0\rangle \in C \wedge \langle X_{lst}, END\rangle \in C \\ & \wedge \bigwedge_{i=1}^{n-1} succ(X_i, X_{i+1}, L) \wedge \bigwedge_{i=1}^{n} \langle X_i, S_i\rangle \in C \\ & \wedge \forall u(X_n \subsetneq u \subsetneq X_{lst} \Rightarrow \langle u, BLANK\rangle \in C) \\ & \wedge \forall u, v, w(\langle u, v\rangle \in C \wedge \langle u, w\rangle \in C \Rightarrow v = w). \end{aligned} \tag{11}$$

The last two universal subformulas in (11) say that the input is padded with blanks and that C is a "function", i.e., every tape symbol is *uniquely* defined.

Now we can write the subformula $I(R)$ of (6) as follows:

$$I(R^{t+5}) \equiv_{df} \exists X^{t+2}, Y^{t+2}\Big(IC(X) \wedge \langle X, Y\rangle \in R\Big). \tag{12}$$

Note that by (5) the type of $\langle X, Y\rangle$ in (12) is $t+4$; hence the type of R is $t+5$, since types in atomic formulas of Ω should differ by one: $x^k \in y^{k+1}$.

The only drawback of the formula (11) (consequently, of (12)) is that it is *superlinear* in the length of input n. The reason is that we introduced $O(n)$ variables $X_1, \ldots, X_n$ to index the sequence of input bits. Even if we are using the economic binary notation for variable indexes, it gives length increase of order $n\log(n)$. Even worse, since in the verbose fully typed version of Ω we must annotate all variable occurrences with their types, and the type $t-1$ in (11) is of length *linear in the size of input* (even written in binary), the formula (11) with all variables types written explicitly is of length $O(n^2)$.

Thus the best lower bound for the verbose fully typed Ω we can get with the initial formula (11)–(12) is (using (A.5) in Corollary A.1)

$$\exp_\infty(\exp(\sqrt{cn})).$$

Still, this is a superpolynomially growing stack of twos.

For the succinct minimally typed Ω we can get with the initial formula (11) a *stronger* lower bound of the form (using (A.4) of Corollary A.1)

$$\exp_\infty(\exp(cn/\log(n))).$$

In Section 17 we describe a more economic way to represent an input. Nevertheless, the solution with the initial formula (11) we suggested here is very simple and intuitive. Also, most importantly, it gives the lower bounds for sentences of Ω of *fixed quantifier prefix complexity*; see Section 16.

14. Final ID, subformula $F(R)$

Analogously, the accepting ID $\$q_a b \ldots b\$$ is specified by:

$$\begin{aligned} FC(C^{t+2}) \equiv_{df}\ & \langle X_{fst}, END\rangle \in C \wedge \langle X_{lst}, END\rangle \in C \wedge \langle X_0, Q_a\rangle \in C \\ & \wedge \forall u(X_1 \subseteq u \subsetneq X_{lst} \Rightarrow \langle u, BLANK\rangle \in C) \\ & \wedge \forall u, v, w(\langle u, v\rangle \in C \wedge \langle u, w\rangle \in C \Rightarrow v = w). \end{aligned}$$

Now we can write the subformula $F(R)$ of (6):

$$F(R^{t+5}) \equiv_{df} \exists X^{t+2}, Y\Big(FC(X\ ) \wedge \langle Y, X\rangle \in R\Big). \tag{13}$$

Note that both formulas FC, F are *quasi-fixed.*

15. Correct transitions, subformula $C(R)$

The following lemma due to Stockmeyer [13, Lemma 2.14, p. 38], is a basic tool for arithmetization of Turing machines. It allows one to check, for a given TM M and two IDs $\mathbf{d}_1$ and $\mathbf{d}_2$, whether $\mathbf{d}_2$ results from $\mathbf{d}_1$ by one step of M (symbolically $\mathbf{d_2} \in Next_M(\mathbf{d_1})$) only by making *local checks.* Such a local check consists in comparing the $(j-1)$th, jth, and $j+1$th symbols of $\mathbf{d_1}$ and $\mathbf{d_2}$ for all j. One has $\mathbf{d_2} \in Next_M(\mathbf{d_1})$ if and only if all such local checks succeed.

Lemma 7 (L. Stockmeyer). *Let M be any TM with tape alphabet Σ and set of states $\mathcal{Q}$. Suppose $\$ \notin \Sigma \cup \mathcal{Q}$ is the tape end marker, $b \notin \Sigma \cup \mathcal{Q}$ is a blank, and $\Delta = \Sigma \cup \mathcal{Q} \cup \{\$, b\}$. There exists a function $N_M : \Delta^3 \to \mathcal{P}(\Delta^3)$ satisfying the following property*:

for any two IDs $\mathbf{d}_1, \mathbf{d}_2$ of M such that

$$\$\mathbf{d}_1\$ = d_{1,0}\ d_{1,1} \ldots d_{1,j-1}\ d_{1,j}\ d_{1,j+1} \ldots d_{1,k}\ d_{1,k+1}$$
$$\$\mathbf{d}_2\$ = d_{2,0}\ d_{2,1} \ldots d_{2,j-1}\ d_{2,j}\ d_{2,j+1} \ldots d_{2,k}\ d_{2,k+1}$$

one has $\mathbf{d_2} \in Next_M(\mathbf{d_1})$ iff $d_{2,j-1}d_{2,j}d_{2,j+1} \in N_M(d_{1,j-1}d_{1,j}d_{1,j+1})$ for all $j \in \{1, \ldots, k\}$. □

Note that the graph of any function N_M in Lemma 7 is *constant* since it depends only on $card(\Delta)$, fixed when we choose a TM M, before we start constructing (6). This graph may be defined by the following boolean formula

$$\Psi_M(x,y,z,x',y',z') \equiv_{df} \bigwedge_{s_1s_2s_3 \in \Delta^3} \Big(x = s_1 \wedge y = s_2 \wedge z = s_3 \Rightarrow \bigvee_{s_1's_2's_3' \in N_M(s_1s_2s_3)} (x' = s_1' \wedge y' = s_2' \wedge z' = s_3') \Big).$$

The size of this formula (with types erased) is constant once the description of M is fixed. However, the *fixed* number of variable occurrences in Ψ_M are annotated with types linearly depending on the length of input. Hence, Ψ_M is quasi-fixed, and its size is linear in the size of input.

We are now ready to write the formula $C(R)$ of (6):

$$\begin{aligned} C(R) \equiv_{df} \forall X, Y \Big(\langle X, Y\rangle \in R^{t+5} \Rightarrow \\ \forall x,y,z,a,b,c,a',b',c' \Big[adj3(x,y,z,L) \\ \wedge \langle x,a\rangle \in X \wedge \langle y,b\rangle \in X \wedge \langle z,c\rangle \in X \\ \wedge \langle x,a'\rangle \in Y \wedge \langle y,b'\rangle \in Y \wedge \langle z,c'\rangle \in Y \\ \Rightarrow \Psi_M(a,b,c,a',b',c') \Big] \\ \wedge \Big[\neg \exists x (\langle x, Q_a\rangle \in Y \Rightarrow \exists Z \langle Y, Z\rangle \in R \Big] \Big). \end{aligned} \tag{14}$$

(Recall that Q_a is a variable of type $t-1$ corresponding to the accepting state $q_a \in \mathcal{Q}$.)

This finishes the definition of the sentence (6) expressing the fact that a given $\exp_\infty(\exp(n)) - 2$-space bounded TM M accepts an input x of length n.

16. Lower bounds for Ω with fixed quantifier prefix

The subformulas A, C, F of $\Phi_{M,x}$, defined by (6), (10), (13), (14), are *quasi-fixed*, hence, *linearly bounded* in the length of input n. The initial subformula I of $\Phi_{M,x}$ defined by (12), (11) is of size $O(n^2)$ for the verbose fully typed Ω, of size $O(n \log(n))$ for the succinct partially typed Ω, and the number

of quantifiers in *I does not depend* on n. Therefore, we may precisely state the first lower bounds for Ω we just obtained:

Theorem 8 (Fixed quantifier prefix lower bounds)**.** *There exists a finite fixed quantifier prefix QP such that any decision algorithm for* Ω *requires space exceeding, respectively*:
(*For fully typed* Ω)

$$\exp_\infty(\exp(c\sqrt{|\Phi|})), \tag{15}$$

for some constant $c > 0$ *and infinitely many prenex sentences* Φ *of verbose* Ω *with quantifier prefix QP*;
(*For partially typed* Ω)

$$\exp_\infty(\exp(c|\Phi|/\log(|\Phi|))), \tag{16}$$

for some constant $c > 0$ *and infinitely many prenex sentences* Φ *of succinct* Ω *with quantifier prefix QP.* □

Note that already (15), (16) provide a *superpolynomial rate of stack of 2's growth* in the lower bounds for both versions of Ω.

In the remainder of the paper we describe a more economic method for representing an input. The solution with formula (11) (the only one non-quasi-fixed) we suggested here is very simple and intuitive. Also, most importantly, it gives the lower bounds for sentences of Ω of *fixed quantifier prefix complexity*. This is not the case for an alternative solution we suggest below. However, we will push (15) up to $\exp_\infty(\exp(c|\Phi|/\log(|\Phi|)))$ and (16) to $\exp_\infty(\exp(cn))$.

17. More succinct initial formula

The non-quasi-fixed initial formula (11)–(12) was constructed by using $O(n)$ variables, n is the length of input. This non-economic representation led to non-optimal lower bounds of Theorem 8. In this section we describe a more clever way to represent an input by using only logarithmically many variables. We split the job into two subtasks. First, in Section 17.1, we describe a method to represent an input by a formula linear in the length of input. Second, in Sections 17.2–17.4, we describe how to "copy" the input represented that way onto the initial ID of a TM.

17.1. Input formula

Let an input $s_1 \dots s_n \in \{0,1\}^+$ of length n be given, padded by blanks to the length 2^m with $m = \lceil \log(n) \rceil$, if necessary. We will show how to construct the formula $INPUT_m(d_m, d_{m-1}, \dots, d_2, d_1, x)$, with variables d_i of type 0 and x of type 1, of size $O(n)$ with the property:

> when $d_m, d_{m-1}, \dots, d_2, d_1$ are assigned binary values 0, 1, the formula $INPUT_m(d_m, d_{m-1}, \dots, d_2, d_1, x)$ is true iff $x = s_k$, where $k = 1+$ the number represented in binary by $d_m d_{m-1} \dots d_2 d_1$.

(x is of type 1 because type 0 has just two values, insufficient to represent the third value "blank" in the padded input.)

To write $INPUT_m$ we use auxiliary formulas $input_{i,j}$ defined inductively:

$$\begin{aligned} input_{0,j}\ (x) &\equiv_{df} x = s_{j+1}(\text{for } 0 \leqslant j < 2^m),\\ input_{i+1,j}\ (d_{i+1}, d_i, \ldots, d_1, x) &\equiv_{df}\\ &(d_{i+1} = 0 \Rightarrow input_{i,j}(d_i, \ldots, d_1, x)) \wedge\\ &(d_{i+1} = 1 \Rightarrow input_{i,2^i+j}(d_i, \ldots, d_1, x)). \end{aligned}$$

Intuitively, $input_{i,j}$ describes the segment of the input of length 2^i starting from position $j+1$.

It remains to define

$$INPUT_m(d_m, d_{m-1}, \ldots, d_2, d_1, x) \equiv_{df} input_{m,0}(d_m, d_{m-1}, \ldots, d_2, d_1, x).$$

In this formula the variable d_i appears 2^{m-i+1} times. Even if we write the indexes of d_is in *unary*, the total space occupied by these indexes in $INPUT_m$ will be equal

$$\sum_{k=1}^{m} k \cdot 2^{m-k+1} = 4 \cdot 2^m - 2m - 4 < 4 \cdot 2^m = \mathrm{O}(n),$$

which may be easily shown by induction or using Maple. The formula $INPUT_m$ additionally contains the linear number of occurrences of x (indexed with index 0, which occupies constant space written in unary), d, 0, 1, parentheses and logical signs, each one of constant size. Note also that all variables in $INPUT_m$ have fixed types independent of input length. Therefore, the total size of $INPUT_m$ is $\mathrm{O}(n)$, as needed, both in succinct and fully typed versions of Ω. Clearly, the formula $INPUT_m$ can be constructed in polynomial time.

The complications above are caused by the fact that writing straightforwardly

$$INPUT_m(d_m, d_{m-1}, \ldots, d_2, d_1, x) \equiv \bigwedge_{b_i \in \{0,1\}} d_m = b_m \wedge \ldots \wedge d_1 = b_1 \Rightarrow x = s_{b_m \ldots b_1},$$

would result in a formula of superlinear size, since each variable appears n times and there are $\mathrm{O}(\log(n))$ different variables. Thus the formula grows at least as $\mathrm{O}(n \log(\log(n)))$, faster than we can afford.

17.2. Counting long distances in a chain

To write the initial formula (11), (12) we need to "copy" an arbitrary input string $s_1 \ldots s_n$, represented by the formula $INPUT_m$ on the initial tape, saying that the 3rd, …, $(n+2)$nd symbols of the initial ID of the TM M equal *ZERO* or *ONE*, corresponding to $s_i = 0$ or $s_i = 1$. The straightforward method of Section 13 results in a superlinear size formula. In order to address n successors in a chain L more economically, we will define the formulas

$$SUCC_m(X_1, \underbrace{d_m, \ldots, d_1}_{m};\ Y, \underbrace{e_m, \ldots, e_1}_{m})$$

for $m = \lceil \log(n) \rceil$, such that d_i's, e_i's take binary bits 0, 1 as values and

when the sequences $d_m \dots d_1$ and $e_m \dots e_1$ are considered as binary representations for the natural numbers n_1, n_2, respectively, then Y is the $(n_2 - n_1)$th successor of X_1 in the chain L^t (with respect to the $succ(U, V, L)$ relation (9)), provided $n_2 \geqslant n_1$.

This gives a succinct way to count distances up to $2^m - 1$ between elements in the chain L^t and thus to address remote successors (up to $2^m - 1$st) of X_1 without the $O(n^2)$ blow-up. With formulas $INPUT_m$ and $SUCC_m$ we can succinctly define that the initial tape C contains a subset of $L \times L$, where 2^m (with $m = \lceil \log(n) \rceil$) successive elements in L^t starting with X_1 index input values $s_1, \dots, s_{2^m}$ as follows:

$$D(X_1) \equiv_{df} \forall d_m, \dots, d_1, v, Y, V \Big(INPUT_m(d_m, \dots, d_1, v) \wedge SUCC_m(X_1, \underbrace{0, \dots, 0}_{m};\ Y, d_m, \dots, d_1)$$
$$\Rightarrow \Big[(v = \{0\} \wedge V = ZERO) \vee (v = \{1\} \wedge V = ONE) \vee (v = \{0,1\} \wedge V = BLANK)$$
$$\Rightarrow \langle Y, V \rangle \in C \Big] \Big),$$

where Y, V are of type $t - 1$, v of type 1, and all other variables of type 0. The variable v represents three possibilities in the input: $\{1\}$ for 1, $\{0\}$ for 0, and $\{0,1\}$ for the blank (padding inputs to length $2^{\lceil \log(n) \rceil}$).

Therefore, the subformula IC of the initial formula (12) may be defined more economically than (11) as follows:

$$\begin{aligned} IC(C^{t+2}) \equiv_{df} & \langle X_{fst}, END \rangle \in C \wedge \langle X_0, Q_0 \rangle \in C \wedge \langle X_{lst}, END \rangle \in C \wedge D(X_1) \wedge \\ & \forall YZ(SUCC_m(X_1, 0, \dots, 0;\ Y, 1, \dots, 1) \wedge Y \subsetneq Z \subsetneq X_{lst} \\ & \Rightarrow \langle Z, BLANK \rangle \in C) \wedge \forall u, v, w(\langle u, v \rangle \in C \wedge \langle u, w \rangle \in C \Rightarrow v = w). \end{aligned} \tag{17}$$

Before we start defining $SUCC$, let us explicitly define the auxiliary relations $<$ and $\leqslant$ on elements of type 0 as follows:

$$x^0 < y^0 \equiv_{df} x \in \{0\} \wedge y \in \{1\},$$
$$x^0 \leqslant y^0 \equiv_{df} \neg y < x.$$

The formula $SUCC_m$ is defined by induction on $i = 0, \dots, m$, similarly to the inductive definition of $INPUT_m$. As the base case let

$$SUCC_0(X^{t-1}, d_1;\ Y^{t-1}, e_1) \equiv_{df} (d_1 \leqslant e_1) \wedge (d_1 = e_1 \Rightarrow X = Y) \wedge (d_1 < e_1 \Rightarrow succ(X, Y, L^t)). \tag{18}$$

For $i \geqslant 0$ define, inductively:

$$\begin{aligned} & SUCC_{i+1}(X^{t-1}, d_{i+1}, d_i, \dots, d_1;\ Y^{t-1}, e_{i+1}, e_i, \dots, e_1) \equiv_{df} (d_{i+1} \leqslant e_{i+1}) \\ & \quad \wedge (d_{i+1} = e_{i+1} \Rightarrow SUCC_i(X, d_i, \dots, d_1;\ Y, e_i, \dots, e_1)) \\ & \quad \wedge \Big(d_{i+1} < e_{i+1} \Rightarrow \exists Z_1^{t-1}, Z_2 \Big[succ(Z_1, Z_2, L^t) \wedge SUCC_i(X, d_i, \dots, d_1;\ Z_1, \underbrace{1, \dots, 1}_{i}) \\ & \quad \wedge SUCC_i(Z_2, \underbrace{0, \dots, 0}_{i};\ Y, e_i, \dots, e_1) \Big] \Big). \end{aligned} \tag{19}$$

Clearly (by induction), $SUCC_m$ defined by (18), (19) allows us to count distances up to 2^m between elements of the chain L^t. The drawback of the definition (19) is that $SUCC_m(X^{t-1}, d_m, \ldots, d_1; Y^{t-1}, e_m, \ldots, e_1)$, fully expanded by using (18) and (19) to a formula without occurrences of $SUCC$, will contain $O(2^m) = O(n)$ occurrences of variables X, Y. This is easy to see: if $SUCC_i(X, \ldots)$ contains k occurrences of X after full expansion, then $SUCC_{i+1}(X, \ldots)$ will contain $2k$ such occurrences. Thus, we do not gain anything with definition (19), as compared with the straightforward method with n new variables described in Section 13. However, we can do better, as shown in the next section.

17.3. Abbreviation trick

The right-hand-side of (19) defines $SUCC_{i+1}$ by using 2 occurrences of $SUCC_i$. This may be written in an equivalent more economic way with just one occurrence of $SUCC_i$, by applying a well-known abbreviation trick due to Fischer–Meyer–Rabin–Stockmeyer. To keep the paper self-contained, Appendix A.2, sketches a proof (similar to [2, Ch. 7, Lemma 3], [1, Sec. 3, Theorem 3.1]) of the simple case, sufficient for our purposes, when all multiple subformula occurrences are *positive* (as $SUCC_i$ in (19)). As an advantage we do not need the equivalence connective $\Leftrightarrow$.

Lemma 9. *Given a quantifier-free formula Φ containing m positive occurrences $P(\bar{t}_i)$ (for $i = 1, \ldots, m$) of the same predicate P with different parameters $\bar{t}_i$, and no negative occurrences of P, one can construct in polynomial time an equivalent formula Δ of size $O(|\Phi|)$, containing just one positive and no negative occurrences of P. More precisely, the formula Δ is*:

$$\exists x_1 y_1 \ldots x_m y_m \Big(\Phi' \wedge \forall u v \bar{z} \Big[\bigvee_{i=1}^{m} (u = x_i \wedge v = y_i \wedge \bar{z} = \bar{t}_i) \Rightarrow (u = v \Rightarrow P(\bar{z})) \Big] \Big), \tag{20}$$

where $x_1, y_1, \ldots, x_m, y_m, u, v, \bar{z} = z_1, \ldots, z_{arity\ (P)}$ are fresh variables, and

$$\Phi' \equiv \Phi \Big[x_i = y_i / P(\bar{t}_i) \Big]_{i=1}^{m}.$$

17.4. Complexity of SUCC

Applying Lemma 9, and putting all quantifiers in front of the formula, we can rewrite the definition (19) of $SUCC_{i+1}$ equivalently by using just one occurrence of $SUCC_i$ as follows:

$$\begin{aligned}
&SUCC_{i+1}(X^{t-1}, d_{i+1}, d_i, \ldots, d_1;\ Y^{t-1}, e_{i+1}, e_i, \ldots, e_1) \equiv_{df} \exists Z_1, Z_2, x_1, y_1, x_2, y_2, x_3, y_3 \\
&\forall u, v, Z, z_1, \ldots, z_i, Z', z'_1, \ldots, z'_i \Big\{ (d_{i+1} \leqslant e_{i+1}) \wedge (d_{i+1} = e_{i+1} \Rightarrow x_1 = y_1) \\
&\wedge (d_{i+1} < e_{i+1} \Rightarrow x_2 = y_2 \wedge x_3 = y_3 \wedge succ(Z_1, Z_2, L^t)) \\
&\wedge \Big[(u = x_1 \wedge v = y_1 \wedge Z = X \wedge Z' = Y \wedge \textstyle\bigwedge_{j=1}^{i} (z_j = d_j \wedge z'_j = e_j)) \\
&\quad \vee (u = x_2 \wedge v = y_2 \wedge Z = X \wedge Z' = Z_1 \wedge \textstyle\bigwedge_{j=1}^{i} (z_j = d_j \wedge z'_j = 1)) \\
&\quad \vee (u = x_3 \wedge v = y_3 \wedge Z = Z_2 \wedge Z' = Y \wedge \textstyle\bigwedge_{j=1}^{i} (z_j = 0 \wedge z'_j = e_j))
\end{aligned} \tag{21}$$

$$\Rightarrow \Big(v = u \Rightarrow SUCC_i(Z, z_i, \ldots, z_1;\ Z', z'_i, \ldots, z'_1)\Big)\Big]\Big\}.$$

Therefore, each iteration expanding $SUCC_{i+1}$ via $SUCC_i$ using (21) introduces:
(1) four variables, Z_1, Z_2, Z, Z', of type $t-1$,
(2) eight variables, $x_1, y_1, x_2, y_2, x_3, y_3, u, v$, of type 0,
(3) $2i$ variables, $z_1, \ldots, z_i, z'_1, \ldots, z'_i$, of type 0.
Consequently, m iterations reducing $SUCC_m$ to a formula without occurrences of $SUCC$ will introduce:
(1) $O(m)$ new quantified variables of type $t-1$,
(2) m occurrences of the variable L of type t,
(3) $O(m^2)$ variables of type 0.

By definition (21) of $SUCC$, the full expansion of $SUCC_m$, into a formula without occurrences of $SUCC$, will contain:
(1) $O(m)$ occurrences of variables of type $t-1$,
(2) $O(m^2)$ occurrences of variables of type 0, boolean connectives and parentheses.
Since $m = \lceil \log(n) \rceil$, if we ignore the types of variables, the length of $SUCC_m$, after full expansion, will be $O(m^2 \log(m^2)) = O(\log^2(n) \cdot \log(\log^2(n)))$, since we need $\log(k)$ bits for indexes to represent k different variables. This is $O(n)$, and thus leads to a linearly bounded initial formula (17) in the case of succinct, partially typed Ω. However, for the fully typed Ω, each of the $O(m) = O(\log(n))$ occurrences of variables of types $t-1, t$ should be annotated with types of length $O(n)$. Therefore, the formulas $SUCC_m$ and (17) are of superlinear length $O(n \log(n))$ in the case of verbose fully typed Ω. The more numerous $O(m^2)$ variables of type 0 do not contribute to this superlinear length increase, because their full type annotations take only $O(m^2) = O(\log^2(n))$ bits.

Note that this superlinear explosion does not occur in the first-order theories, which do not require variable type annotations.

The $O(n \log(n))$ *superlinear explosion* takes place only for the *verbose* version of Ω, which requires all variable occurrences to be annotated with types. For the *succinct* version of Ω, which requires only a minimal information about variable types allowing for an unambiguous full type annotation, the order of growth for the formula $SUCC_m$ is $O(m^2 \log(m^2)) = O(\log^2(n) \cdot \log(\log^2(n)))$, i.e., is *sub-linear*, and the reduction taking an input of length n into the formula $SUCC_m$, with $m = \lceil \log(n) \rceil$, is *linearly bounded.*

18. Stronger lower bounds: Main theorem

The initial formula $I(R)$ of (6) defined by (11), (12) was the only non-quasi-fixed and non-linearly bounded formula in the construction preceding Section 16. In Section 17 we constructed a more succinct initial formula $I(R)$ of size $O(n \log(n))$ in the case of fully typed Ω, and size $O(n)$ in the case of partially typed Ω. Therefore, Lemma A.1 and Corollary A.1 apply, and we obtain our main result.

Theorem 10 (Lower bounds for Ω)**.** *Every decision algorithm for* Ω *requires space* (*hence time*) *exceeding*:

(1) $\exp_\infty(\exp(c \cdot |\Phi|/\log(|\Phi|)))$ *for some constant* $c > 0$ *and infinitely many sentences* Φ *of verbose* Ω;
(2) $\exp_\infty(\exp(c \cdot |\Phi|))$ *for some constant* $c > 0$ *and infinitely many sentences* Φ *of succinct* Ω. □

19. Concluding remarks

On inductive definitions. We succeeded to construct the generic reduction *without using inductive definitions* to define large linearly ordered sets in Ω. Such definitions are usually necessary in lower bounds proofs. Inductive definitions are only used in Section 17 to write a more succinct initial formula representing an input. This is a big advantage, because otherwise:

(1) The best lower bound we could obtain for fully typed Ω would be only $\exp_\infty(\exp(\sqrt{cn}))$ instead of $\exp_\infty(\exp(cn/\log(n)))$. Indeed, expanding inductive definitions and using the well-known abbreviation trick due to Fischer–Meyer–Rabin–Stockmeyer (so as to avoid exponential blow-ups), one gets formulas with *linearly many variable occurrences*. Since in fully typed Ω variable occurrences are annotated with types, which may linearly depend on the length of input, using inductive definitions would necessarily lead to non-linear (quadratic) reductions, and thus to weaker lower bounds.
(2) We would fail to have the fixed quantifier prefix complexity results of Theorem 8. The formula $\Phi_{M,x}$ in (6) we construct before Section 16 to provide a reduction from $A \in DSPACE(\exp_\infty(\exp(n)))$ to Ω has a *fixed number of quantifiers and quantifier alternations*, independent of A, which yields a *fixed quantifier prefix* lower bound complexity. This quite unusual result should be contrasted to the results of [6], which needs more and more quantifier alternations to get increase in complexity.

On finite axiomatizability. The theory Ω was defined *semantically* and is not finitely axiomatizable. Solomon Feferman asked (LICS'97) whether this non-finite axiomatizability is really essential. Although the proof presented here does not give a direct answer, returning to the original proof presented in [17] based on the uniform lower bound method due to Compton and Henson [1], we may now respond by:

> Any finitely axiomatizable subtheory of Ω (in the same language) has the same space lower bound $\exp_\infty(\exp(dn))$ for some constant $d > 0$.

This is because [1] spends extra effort on proving stronger *inseparability* results, which imply lower bounds not only for theories, but for all their subtheories.

Upper bound for Ω. Since we have not used the full power of inductive definitions in settling the lower bounds for Ω, it might seem challenging to push these bounds even higher. However, this is *impossible*. In fact, the maximal size of a variable type in a formula of Ω of length n is $O(n)$. Therefore, all quantified variables in a sentence of Ω run over finite domains $\mathcal{D}_{2^{O(n)}}$ of size at most $\exp_\infty(\exp(O(n)))$. Obviously, this space is enough for a decision procedure.

Any "more nonelementary" theories? The following challenging problem in [1, Problem 10.12] is open/closed (modulo what is considered "natural"):

> Is there a "natural" decidable theory, which is not primitive recursive?

In [17] we constructed several (pathological) variants of Ω of arbitrary complexity. After all, expressiveness of Ω is based on ability to write types of variables in binary. Therefore, it suffices to use any other, more expressive, *non-primitive recursive notation for types*, instead of binary.

Other candidates may be looked among logical counterparts of the higher-order polymorphic lambda calculi in the same way as Ω corresponds to the simply typed lambda calculus. Of course, it is questionable whether these theories may be considered "natural," and whether they may be kept decidable. Urquhart [16] settles nonprimitive-recursive lower bounds for relevance logics.

Higher lower bound for fully typed Ω. It remains open whether the lower bound for the fully typed Ω can be improved from $\exp_\infty(\exp(cn/\log(n)))$ to $\exp_\infty(\exp(cn))$. Recall that the only size $O(n\log(n))$ and non-linearly bounded formula we used was *SUCC* in Section 17.4 for "copying" a sequence of input bits onto the initial ID. Is there any mean to do it by an $O(n)$ fully typed formula?

Acknowledgments

The author is grateful to David Basin, Solomon Feferman, Harald Ganzinger, Paweł Urzyczyn, Moshe Vardi, and anonymous referees for insightful comments, discussions, and remarks.

Appendix A. Lower bound details

A.1. Generic reductions and reductions via length order

This section provides a succinct exposition of essential lower bound tools used in the body of the paper, including the generic reduction and the order of reduction. If every problem in a class $\mathcal{C}$ is reducible to a problem T, then T is approximately as complex as any problem in $\mathcal{C}$, modulo the order of reduction. This is summarized by Lemma A.1 and Corollary A.1.

Space hierarchy theorem. We found it most convenient to work with space complexity classes. However, all the arguments below may be appropriately modified and carried out for (non)deterministic time complexity classes. In fact, there is only a slight difference in claiming that a problem requires space or (non)deterministic time exceeding $\exp_\infty(\exp(dn))$ for some $d > 0$ infinitely often. Note that the space claim is the strongest.

A function $S(n) > \log_2(n)$ is called *space constructible* if there exists an $S(n)$-space bounded TM M such that for each n there exists an input of length n on which M actually uses $S(n)$ tape cells. If for all n, M uses exactly $S(n)$ cells on any input of length n, then $S(n)$ is said *fully space constructible*. Any space constructible $S(n)$ is fully space constructible, [4, p. 297]. It is a routine exercise to show that functions like $\exp_\infty(\exp(n)) - k$ (with $k \in \omega$) and $\exp_\infty(\exp(n/2))$ are (fully) space constructible. Throughout the paper we assume fully space constructible functions.

As usual, $DSPACE(S(n))$ denotes the class of languages recognized by the $S(n)$-space bounded deterministic Turing machines. To settle the space lower bounds, we rely on the following well-known separation result; see, e.g., [4, Theorem 12.8, p. 297]:

Theorem A.1. *Let functions* $S_1(n)$ *and* $S_2(n)$ *be such that* $\lim_{n\to\infty} S_1(n)/S_2(n) = 0$, *and* $S_1(n), S_2(n)$ *are each at least* $\log_2(n)$. *Then there is a language in*

$$DSPACE(S_2(n)) \setminus DSPACE(S_1(n)).$$

We use both linearly bounded and nonlinearly bounded deterministic time polynomial reducibilities in conjunction with Theorem A.1 to settle the space lower bounds by *generic reduction*; cf. [2,7,12–15]:

The following lemma explains the method of proving lower bounds by generic reduction. If a class of problems is reducible to a problem, then the problem is as difficult as an "average" problem is the class, modulo the order of reducibility.

Lemma A.2. (Lower bounds by generic reduction). *Let*:

(1) g and h *be functions such that for every constant* $c_1 > 0$ *there exists a constant* $c_2 > 0$ *such that for all* $n \in \omega$ (except, maybe, finitely many) *one has*

$$h(c_1 \cdot g(n)) \leqslant c_2 \cdot n, \tag{A.1}$$

(2) $S(n) \geqslant \exp(n)$ *be fully space constructible, such that for every constants* $c, d > 0$ *the function* $S(dh(cg(n)))$ *is monotone and grows faster than any polynomial,*

(3) T *be a problem such that every problem* $A \in DSPACE(S(n) - 2)$ *is reducible to* T *via length order* $g(n)$.

Then for some $d > 0$ *one has*

$$T \notin DSPACE(S(dh(n))). \tag{A.2}$$

Equivalently, T *requires deterministic space exceeding* $S(h(dn))$ *infinitely often.*

Proof. By Theorem A.1, there is a problem

$$A \in DSPACE(S(n) - 2) \setminus DSPACE(S(n/2)).$$

Since A is reducible to T via length order $g(n)$, for every constant $d > 0$ we have the following chain:

$$\begin{aligned} T \in DSPACE(S(dh(n))) &\Rightarrow A \in DSPACE(S(dh(cg(n))) + p(n)) \\ &\Rightarrow A \in DSPACE(S(dh(c_1 g(n)))) \\ &\Rightarrow A \in DSPACE(S(dc_2 n)), \end{aligned}$$

where $p(n)$ is a polynomial (time necessary to compute a reduction from A to T), c is a constant from (4), and c_1 is a constant slightly larger than c; by assumption, $S(dh(cg(n)))$ grows faster than $p(n)$, and we use the assumption (A.1).

The contrapositive of the above implication chain is

$$A \notin DSPACE(S(dc_2 n)) \Rightarrow T \notin DSPACE(S(dh(n))).$$

Since $A \notin DSPACE(S(n/2))$, it suffices to select $d = 1/2c_2$ to obtain (A.2). □

Remark A.3. The "length order condition" (4) is really important. Deterministic polynomial time computability of reduction is unnecessarily strong, and we use it only following the common practice. In fact, any reduction computable in space $o(S(dh(cg(n))))$ would be appropriate.

Corollary A.4. *Lemma A.*1 *applies for the function* $S(n) = \exp_\infty(\exp(n))$ *and the following reducibilities.*

Order n (linear) reducibility: $g(n) = n$; *in this case* $h(n) = n$ *and*

$$T \notin DSPACE(\exp_\infty(\exp(dn))). \tag{A.3}$$

Order $n\log(n)$ *reducibility*: $g(n) = n\log(n)$; *in this case* $h(n) = n/\log(n)$ *and*

$$T \notin DSPACE(\exp_\infty(\exp(dn/\log(n)))). \tag{A.4}$$

Order n^2 *reducibility*: $g(n) = n^2$; *in this case* $h(n) = \sqrt{n}$ *and*

$$T \notin DSPACE(\exp_\infty(\exp(d\sqrt{n}))). \tag{A.5}$$

Thus, "more tight" reducibilities yield stronger lower bounds. This last corollary is used several times in the main part of the paper.

A.2. Abbreviation trick: Proof of lemma 9

Take fresh variables $x_1, y_1, \ldots, x_m, y_m$ and consider the formula

$$\Phi' \equiv \Phi\Big[x_i = y_i/P(\bar{t}_i)\Big]_{i=1}^{m},$$

obtained from Φ by replacing the ith occurrence of $P(\bar{t}_i)$ with equality $x_i = y_i$. Note that $m = \mathrm{O}(n/\log(n))$, where n is the length of Φ. Thus introducing $2m$ new variables does not lead to more than a linear length increase, because each variable may be represented using $\mathrm{O}(\log(n))$ bits.

Let us show that Φ is equivalent to

$$\Psi \equiv \exists x_1 y_1 \ldots x_m y_m \left(\Phi' \wedge \bigwedge_{i=1}^{m} (x_i = y_i \Rightarrow P(\bar{t}_i)) \right).$$

We must prove that for every interpretation ν of the free variables of Φ (or, equivalently, of Ψ), $\nu(\Phi)$ is true iff $\nu(\Psi)$ is true.

Let $\nu(\Phi)$ be true. We may choose equal values for x_i, y_i if $P(\bar{t}_i)$ is true, and different values for x_i, y_i otherwise. Then $\nu(\Psi)$ is true.

Suppose $\nu(\Psi)$ is true for some interpretation ν of its free variables. Then for this interpretation and some values of $x_1, y_1, \ldots, x_m, y_m$ the following subformulas of Ψ:

$$\Phi' \equiv \Phi\Big[x_i = y_i/P(\bar{t}_i)\Big]_{i=1}^{m},$$
$$\psi \equiv \bigwedge_{i=1}^{m} (x_i = y_i \Rightarrow P(\bar{t}_i)). \tag{A.6}$$

are true. Since Φ is positive in $P(\bar{t}_i)$, by construction, Φ' is positive in $x_i = y_i$. Therefore, Φ' is monotone in $x_i = y_i$. This and the truth of (A.6) imply that $\Phi'\left[P(\bar{t}_i)/x_i = y_i\right]_{i=1}^{m}$ is true. But this formula coincides with Φ.

The formula Ψ still contains m occurrences of P. Take fresh variables u, v, and $\bar{z}$ (vector of length equal to the arity of P). The subformula ψ of Ψ containing m occurrences of P is equivalent to

$$\delta \equiv \forall u v \bar{z} \left(\bigvee_{i=1}^{m} (u = x_i \wedge v = y_i \wedge \bar{z} = \bar{t}_i) \Rightarrow (u = v \Rightarrow P(\bar{z})) \right),$$

which contains just one occurrence of P. The proof of the equivalence of ψ and δ is routine.

Finally, let Δ be Ψ with the occurrence of ψ replaced by δ. Clearly, Δ is equivalent to Φ and contains just one positive occurrences of P and no negative occurrences of P, as needed. It is clear that Δ may be constructed from Φ in polynomial time and the size of Δ is linearly bounded by the size of Φ. □

References

[1] K.J. Compton, C.W. Henson, A uniform method for proving lower bounds on the computational complexity of logical theories, Ann. Pure Appl. Logic 48 (1990) 1–79.
[2] J. Ferrante, C. W. Rackoff, The computational complexity of logical theories,in: Lect. Notes Math., vol. 718, Springer-Verlag, Berlin, 1979.
[3] L. Henkin, A theory of propositional types, Fundamenta Mathematicæ 52 (1963) 323–344.
[4] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, Reading, MA, 1979.
[5] R. Hull, J. Su, On the expressive power of database queries with intermediate types, J. Comput. Syst. Sci. 43 (1991) 219–267.
[6] G. Kuper, M. Vardi, On the complexity of queries in the logical data model, Theor. Comput. Sci. 116 (1993) 33–57.
[7] H.R. Lewis, Complexity results for classes of quantificational formulas, J. Comput. Syst. Sci. 21 (1980) 317–353.
[8] H. Mairson, A simple proof of a theorem of Statman, Theor. Comput. Sci. 103 (1992) 387–394.
[9] A.R. Meyer, The inherent computational complexity of theories of ordered sets, in: International Congress of Mathematicians, Vancouver, 1974, pp. 477–482.
[10] A.R. Meyer, Weak monadic second-order theory of successor is not elementary-recursive, in: R. Parikh (Ed.), Logic Colloquium: Symposium on Logic Held at Boston, 1972–1973, Vol. 453 of Lect. Notes Math., Springer-Verlag, 1975, pp. 132–154.
[11] R. Statman, The typed λ-calculus is not elementary recursive, Theor. Comput. Sci. 9 (1979) 73–81.
[12] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time: preliminary report, in: 5th Symposium on Theory of Computing, 1973, pp. 1–9.
[13] L.J. Stockmeyer, The complexity of decision problems in automata theory and logic, PhD thesis, MIT Lab for Computer Science, 1974 (Also /MIT/LCS Tech Rep 133).
[14] L.J. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1977) 1–22.
[15] L.J. Stockmeyer, Classifying the computational complexity of problems, J. Symb. Logic 52 (1) (1987) 1–43.
[16] A. Urquhart, The complexity of decision procedures in relevance logic, in: J.M. Dunn, A. Gupta (Eds.), Truth or Consequences: Essays in Honor of Nuel Belnap, Kluwer, 1990, pp. 61–76.

[17] S. Vorobyov, The "hardest" natural decidable theory, in: G. Winskel (Ed.), 12th Annual IEEE Symp. on Logic in Computer Science (LICS'97), 1997, pp. 294–305.
[18] S. Vorobyov, A. Voronkov, Complexity of nonrecursive logic programs with complex values, in: J. Paredaens, L. Colby (Eds.), Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98), 1998, pp. 244–253.