The Most Nonelementary Theory (A Direct Lower Bound Proof)

Sergei Vorobyov

MPI–I–98–2–007 April 1998

Author's Address

Sergei Vorobyov: Max-Planck Institut für Informatik, Im Stadtwald, D-66123, Saarbrücken, Germany, `sv@mpi-sb.mpg.de`, `http://www.mpi-sb.mpg.de/~sv`.

Publication Notes

The main result of this paper was claimed (with incomplete proof based on a different method) in the author's paper 'The "Hardest" Natural Decidable Theory' presented at the 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97), Warsaw, July 1997, pp. 294–305.

(Publication date: April 2, 1998)

Acknowledgements

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

Abstract

We give a direct proof by generic reduction that a decidable rudimentary theory Ω of finite typed sets (Henkin 1963, Meyer 1974, Statman 1979, Mairson 1992) requires space exceeding infinitely often (lower bound)

$$\exp_\infty(\exp(cn)) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\}\, height\ 2^{cn} \qquad \text{for some constant } c > 0.$$

This gives *the highest* currently known lower bound for a decidable logical theory and affirmatively answers to (Compton & Henson 1990, Problem 10.13, p. 75):

> *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*, have form $\exp_\infty(dn)$, with *just linearly growing* stacks of twos.

Originally, the same lower bound for Ω was settled by (Vorobyov 1997) using the powerful uniform lower bounds method due to (Compton & Henson 1990), and probably would never be discovered otherwise. Although very concise, the original proof left a possibility that the method was pushed out of the limits it was originally designed and intended for, and some hidden assumptions were violated. The independent direct proof presented here aims to dispel all doubts.

Keywords

Complexity, nonelementary theory, generic reduction.

Contents

1 Introduction

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

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

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 *S2S* have lower and upper bounds of the form $\exp_\infty(dn)$, with linearly growing stacks of twos[4]; see (Stockmeyer 1974, Ferrante & Rackoff 1979, Stockmeyer 1987, Compton & Henson 1990) for surveys of known results.

The theory we consider in this paper is far more nonelementary.

Type theory Ω is a very rudimentary fragment of the theory of propositional types due to (Henkin 1963), as defined by (Statman 1979).

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

In this paper we directly prove by generic reduction the following

Main Theorem. *Any Turing machine deciding* Ω *requires space, hence, nondeterministic time exceeding*

$$\exp_\infty(\exp(d \cdot |S|)) = 2^{2^{\cdot^{\cdot^{2}}}}\Big\}\textit{height}\ \exp(d\cdot|S|)=2^{d\cdot|S|} \tag{1}$$

[1]A theory (problem) is called *elementary in the sense of Kalmar* iff it can be decided within time (or space) bounded above by a fixed k-story 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 (Meyer 1975) in May 1972.

[4](Meyer 1975) proved a weaker lower bound, with a logarithmically growing stack.

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

(Theorems 18.1, 20.1 below refine the Main Theorem for two different versions of Ω and for fixed quantifier prefixes.)

The lower bound (1) remains the same (with a different constant), no matter which reasonable computational model is used[5].

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, (Meyer 1974, Theorem pp. 478–479, no. 7) claimed the

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

lower bound. (Statman 1979) 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 1992) sketched the proof that Ω is nonelementary, also without any explicit lower bounds (note that Mairson's proof *does not imply* the lower bound (1)). The high complexity of Ω came unnoticed until (Vorobyov 1997) settled, by using the method of (Compton & Henson 1990), the lower bound (1) 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, (Vorobyov 1997) showed that a long-standing, currently still open *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 by (Compton & Henson 1990, Problem 10.11):

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

Recently (Vorobyov & Voronkov 1998) used the lower bound (1) 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 & Vardi 1993) and also (Hull & Su 1991) considered similar formalisms of logical queries over sets with the *powerset* constructor. They

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

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 & Vardi 1993) use *unary* notation for iterated powersets. Consequently, in Ω we need just $O(\log(256))$ bits to say 'x is an element of $\underbrace{\mathit{powerset}(\ldots(\mathit{powerset}}_{256 \text{ times}}(\{0,1\})\ldots)$',
which requires $O(256)$ bits in the formalism of (Kuper & Vardi 1993). This exponential succinctness translates into the exponential speed-up in the growth of stacks of twos. The other (main) reason is that in proving the lower bound for Ω we *(almost) do not need inductive definitions*, whereas (Kuper & Vardi 1993) do need them to define large sets. In the remainder of this section and in Remarks 15.1, 19.2, 21.1 we discuss why inductive definitions lead to *poorer* lower bounds.

Originally, the lower bound (1) in the Main Theorem was discovered by using the powerful uniform lower bounds method due to (Compton & Henson 1990) in October 1996, and, probably, would never be discovered otherwise (recall that it came unnoticed in (Meyer 1974, Statman 1979, Mairson 1992)). Since the lower bound (1) was first reported in (Vorobyov 1997), we felt it 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 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 (Vorobyov 1997).

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 (Compton & Henson 1990), suggests that the lower bound for Ω proved in (Vorobyov 1997) should be degraded to a more modest $\exp_\infty(\exp(\sqrt{cn}))$ (note: still a superlinear 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 (Vorobyov 1997).

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 the lower bounds. In Ω this can be done with a fixed quantifier prefix, with slightly weaker lower bounds. This came unnoticed in (Vorobyov 1997).

Outline. The paper is organized as follows. Sections 2 and 3 describe preliminaries and lower bounds basics. Section 4 presents the proof plan, and the sections that follow implement it. Section 18 makes an intermediate pause by presenting simple lower bounds for a fixed quantifier prefix, and the succeeding sections push up the lower bounds to the strongest possible.

2 Preliminaries

As usual, $\mathcal{P}(X)$ and $card(X)$ denote the set of all subsets of a set X and its cardinality respectively. We assume the basic knowledge concerning words, languages, complexity, reductions, big-Oh notation, and use all standard notation for words, length, etc. By ω we denote 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 $\exp_{m+1}(n) = 2^{\exp_m(n)}$.

Remark 2.1 Note that $\exp_\infty(n) = \exp_n(1)$. Throughout the paper we use $\exp_\infty(f(n))$ as a shorthand for $\exp_\infty(\lfloor f(n) \rfloor)$. $\square$

Type theory Ω is a very rudimentary fragment of the theory of propositional types due to (Henkin 1963), as defined by (Statman 1979, Mairson 1992).

Definition 2.2 (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 as always, by using $\neg$, $\wedge$, and $\forall$.

The interpretation of Ω is as follows: $\mathbf{0}$ denotes 0, $\mathbf{1}$ denotes 1, and x^n ranges over $\mathcal{D}_n$, where $\mathcal{D}_0 = \{0, 1\}$ and $\mathcal{D}_{n+1} = \mathcal{P}(\mathcal{D}_n)$. $\square$

Remark 2.3 Note that $card(\mathcal{D}_i) = \exp_\infty(i+1)$. $\square$

Remark 2.4 Decidability of Ω is *immediate*, because each quantifier runs over a *finite domain.* See Remark 21.3 for the upper complexity bound. $\Box$

Convention. For complexity considerations below 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.* $\Box$

Annotating all variable occurrences in formulas of Ω with their types introduces a big deal of redundancy. 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 the missing type annotations in the last formula may be easily and uniquely reconstructed. Respectively, we define two versions of Ω.

Definition 2.5 We will 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. $\Box$

This distinction becomes important as soon as linear bounded reducibilities are concerned. Consider a conjunction

$$\bigwedge_{i=0}^{p} x_i^k \in Z^{k+1},$$

where the number of conjuncts p is $O(\log(n))$, and the notational length of type k is $O(n)$. Then the length of the conjunction above is $O(n \log(n))$. The same conjunction written in succinct form

$$x_0 \in Z^{k+1} \wedge \bigwedge_{i=1}^{p} x_i \in Z$$

has length $O(n)$. As a consequence, the succinct version of Ω has 'slightly' higher (in fact, nonelementarily higher) lower bounds, as discussed below.

3 Lower Bounds Basics

The main technical tool we need in this section are Lemma 3.3 and Corollary 3.5, which describe the idea of a generic 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. Experts may skip the rest of this section.

The model of computation we use is the ordinary language recognizing deterministic Turing machine M with a semi-infinite (to the right) tape used both for input, work, and output. We may assume without loss of generality that the tape alphabet Σ of M consists of two symbols, $\Sigma = \{0, 1\}$. We 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., (Stockmeyer 1974, Ferrante & Rackoff 1979). The lower bounds we obtain routinely translate to other realistic models of computation, with only different constants.

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.

We need some basic definitions. A function $S(n) > \log_2(n)$ is called *space constructible* iff 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, (Hopcroft, & Ullman 1979, 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.

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 will need the following well-known separation result; see, e.g., (Hopcroft et al. 1979, Theorem 12.8, p. 297):

Theorem 3.1 If $S_2(n)$ is a fully space constructible function,

$$\lim_{n\to\infty} \frac{S_2(n)}{S_1(n)} = 0,$$

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 will use both linearly bounded and non-linearly bounded deterministic time polynomial reducibilities in conjunction with Theorem 3.1 to settle the space lower bounds by *generic reduction*; cf., (Stockmeyer & Meyer 1973, Stockmeyer 1974, Stockmeyer 1977, Ferrante & Rackoff 1979, Lewis 1980, Stockmeyer 1987):

Definition 3.2 (Reducibility 'Via Length Order') Say that a problem A is *polynomial time reducible to a problem B via length order $g(n)$* iff 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:

$$\begin{aligned} x \in A \quad &\Leftrightarrow \quad f(x) \in B, && (2)\\ |f(x)| \quad &\leq \quad c \cdot g(|x|) \ \text{(except, maybe, finitely many } x\text{)}. && (3) \end{aligned}$$

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

In the sequel we will freely speak about 'linearly bounded' formulas meaning that their sizes are linearly bounded by the length of input and they can be constructed in deterministic time polynomial in the length of input.

The following lemma explains the method of proving the 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 3.3 (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

$$h(c_1 \cdot g(n)) \leq c_2 \cdot n, \tag{4}$$

2. $S(n) \geq \exp(n)$ 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

$$T \notin DSPACE(S(dh(n))). \tag{5}$$

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

Proof. By Theorem 3.1, 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), 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 (4).

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 $d = c_2/2$ to obtain (5). □

Remark 3.4 The 'length order condition' (3) 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 3.5 Lemma 3.3 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

$$T \notin DSPACE(\exp_\infty(\exp(dn))). \tag{6}$$

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

$$T \notin \mathit{DSPACE}(\exp_\infty(\exp(d\frac{n}{\log(n)}))). \tag{7}$$

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

$$T \notin \mathit{DSPACE}(\exp_\infty(\exp(d\sqrt{n}))). \tag{8}$$

Thus, 'more economic' reducibilities yield stronger lower bounds.

4 Proof Plan

According to Lemma 3.3 and Corollary 3.5, our aim in the remainder of the paper will be to show that every problem $A \in \mathit{DSPACE}(\exp_\infty(\exp(n)) - 2)$ is reducible to Ω, i.e., there exist a reduction f and a constant c satisfying (2), (3) for an appropriate order $g(n)$ depending on the version of Ω.

Let A be an arbitrary problem in $\mathit{DSPACE}(\exp_\infty(\exp(n)) - 2)$, 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 4.1 Pay special attention to the order of quantifiers in Lemma 3.3: for *every* problem A there *exist* a function f and a constant c. Thus 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 thus may only influence the value of constant c. This is important as soon as linear boundedness is concerned. Otherwise, if we should have considered a description of M as a part of input, the number of pairs of tape symbols would have been *quadratic*. $\square$

We start constructing the sentence $\Phi_{M,x}$ in Section 6, after extending the language of Ω by allowing explicit definitions.

5 Using Explicit Definitions

Let us extend the language of Ω by allowing *explicit definitions*. This extension will lead to simpler and more understandable formulas, but will not really increase the expressive power and complexity of the theory. This is because all explicit definitions can be eventually eliminated from any formula giving only a *linear* blow-up. Thus using explicit definitions will not harm the linear boundedness of reductions we construct. Experts may skip this section.

Set-Theoretic Notions. We will need the usual set-theoretic explicit definitions like

$$x^m \subseteq y^m \quad \equiv_{df} \quad \forall z^{m-1}(z^{m-1} \in x^m \Rightarrow z^{m-1} \in y^m)$$

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:

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

Remark 5.1 1) For notational simplicity we continue to use the term-like notation like $\{x^n\} = y^{n+1}$ instead of a less natural predicate notation *Is-Singleton*(y^{n+1}, x^n). 2) 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. 3) The explicit definitions above are not fully expanded (according to the usual mathematical practice). The full expansion can always be routinely done, giving only a linear increase in the length of formulas. □

Elimination of 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, of course, should be translated 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 routinely done, giving only a *linear* increase in the length of formulas, provided the depth of terms is bounded in advance.

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.

6 Formula for an Accepting Computation

Given an arbitrary but fixed $(\exp_\infty(\exp(n)) - 2)$-space bounded TM M (cf., Remark 4.1) 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

$$\Phi_{M,x} \;\equiv_{df}\; \exists R^{t+5} \left[\begin{array}{l} A \wedge I(R) \wedge C(R) \wedge F(R) \wedge \\ \forall R' \Big(I(R') \wedge C(R') \Rightarrow R \subseteq R' \Big) \end{array} \right], \tag{9}$$

in the language of Ω, where:

1. the variable R^{t+5} stands for a 'run' of M;

2. the type of R is $t + 5$, where t, called the *principal type* (to be defined later in Section 11) *linearly depends* on the input length n;

3. here and below we agree to omit types of some variable occurrences in formulas, they will always be uniquely determined and easy to guess;

4. in fact, the existentially quantified occurrence R^{t+5} in (9) is the *only* variable occurrence needed to be annotated by a type — all other variables of (9) can be uniquely and unambiguously typed;

5. A is an auxiliary formula to be discussed below in Section 14;

6. $I(R)$ says that R contains an initial *instantaneous description* (ID) of M on input x, defined in Sections 15, 19;

7. $C(R)$ says that R is closed with respect to transitions of M, defined in Section 17;

8. the universally quantified subformula in (9) expresses that R is a *minimal* set containing the initial ID and closed with respect to machine's transitions;

9. $F(R)$ says that R contains an accepting ID of M, defined in Section 16;

10. intuitively, the whole formula (9) says that there exists a path from the initial to the final configuration by using transitions of M, or, equivalently, that M accepts x.

7 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 rules of M, and ending with an accepting ID. Since M is an $(\exp_\infty(\exp(n))-2)$-space bounded, we make a standardizing assumption that all its IDs have equal length $\exp_\infty(\exp(n))+1$ and are of the form

$$\$d_1 \dots d_{\exp_\infty(\exp(n))-1}\$, \tag{10}$$

where:

- $\$ \notin \Sigma \cup \mathcal{Q}$ are tape end markers, over which M never tries to come across,
- *exactly one* of d_i's is a head state symbol (we assume that in this case M is in state designated by this state symbol observing the $i+1$-st tape cell), and
- 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 a blank is not in $\Sigma \cup \mathcal{Q}$).

Thus the total (maximal) tape space described by (10) is $\exp_\infty(\exp(n))-2$.

8 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 (9) stipulates that R is a minimal set satisfying these properties.

9 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 a linearly ordered set of cardinality $\exp_\infty(\exp(n))+1$ defined below in Section 10, needed to index the symbols of an ID in (10),

2. $\{x \mid \exists y \langle x, y \rangle \in ID\} = L$ — to represent (10) we need a *total* function with domain of cardinality $\exp_\infty(\exp(n)) + 1$,

3. $card(\{y | \exists x \langle x, y \rangle \in ID\}) = card(\mathcal{Q} \cup \Sigma) + 2$ — we need to represent states from $\mathcal{Q}$, tape symbols Σ, a blank, and the end marker \$ by elements of L.

Thus, an ID (10) is represented an L-indexed sequence of tape symbols (including head state) represented as elements of L, padded by blanks to the length $\exp_\infty(\exp(n)) - 2$, and embraced by \$.

Note that in Ω we can only construct sets of elements of the *same type*; see the definitions of pairs in Section 5. This explains why we have to use subsets of the Cartesian square of L to represent IDs.

10 Large Linearly Ordered Set

Define the predicate 'to be linearly ordered' by (with type t to be defined below in Section 11)

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

and also 'to be a maximal chain' by

$$MC(L^t) \ \equiv_{df} \ LO(L) \wedge \forall L' \Big(LO(L') \Rightarrow L' \subseteq L \Big). \tag{11}$$

Remark 10.1 It is important to notice that we succeeded to define a linear order *without any inductive definitions.* This is one of the reasons Ω is so hard to decide; cf., Remark 21.1. $\Box$

We need the following simple yet useful

Lemma 10.2 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 thus 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. It is easy to see that 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 1. $\Box$

Convention 10.3 Let everywhere below L^t denote a maximal chain, satisfying $MC(L^t)$ defined by (11). $\Box$

11 The Principal Type

By definition, the *principal type* of the formula (9) is t. Recall that the existentially quantified variable R of (9) has type $t + 5$, and that t is the type of the variable L^t denoting a maximal chain (see the previous section).

Section 9 explains why L^t should have cardinality $\exp_\infty(\exp(n)) + 1$, and from Lemma 10.2 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 (9) uniquely defines the principal type t of L^t, as well as (smaller) types of all other variable occurrences in $\Phi_{M,x}$.

12 Successors and Adjacent Triples in a Chain

Define the 'successor' and the 'three adjacent elements' predicates by:

$$\begin{array}{lll} \mathit{succ}(x,y,L^t) & \equiv_{\mathit{df}} & x \in L \wedge y \in L \wedge x \neq y \wedge \\ & & \forall z (x \subseteq z \subseteq y \Rightarrow (z \subseteq x \vee y \subseteq z)), \\ \mathit{adj3}(x,y,u,L^t) & \equiv_{\mathit{df}} & \mathit{succ}(x,y,L) \wedge \mathit{succ}(y,u,L). \end{array} \tag{12}$$

13 Tape, State, and Auxiliary Symbols

We wish to use certain elements of the maximal chain L^t to represent tape, state, and auxiliary symbols, as explained in Section 9. 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 $0, 1 \in \Sigma$[6], and to add conjunctively to the overall formula we construct

$$\bigwedge_{i=1}^{m} v_i \in L^t \;\wedge \bigwedge_{1 \le i \neq j \le m} v_i \neq v_j.$$

(We may assume without loss of generality that L is large enough to possess at least m elements.)

[6]Recall that the number of symbols m is fixed *a priori*, when a TM for a problem is fixed; see Remark 4.1.

The formula above is almost a *fixed* formula depending only on the number of state symbols in the description of M (which is considered a constant; see, Remark 4.1). However, this formula contains a constant number of variable occurrences, each assigned a type *linearly depending* on the input length. Thus the above formula is of linear length.

This situation will repeat several times in the sequel.

Definition 13.1 Call a formula of Ω *quasi-fixed* iff, after erasing all types of variables, it becomes a fixed formula, independent of input. $\square$

Remark 13.2 We will construct, whenever possible, quasi-fixed formulas with variables annotated by types *linearly* depending on input; see Section 11. Thus, the sizes of such quasi-fixed 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 whenever possible. $\square$

14 Auxiliary Formula

Suppose the initial ID of M is $q_0 s_1 \ldots s_n$. 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 (9) is defined as a conjunction

$$\begin{aligned} A \quad \equiv_{df} \quad & \mathit{MC}(L^t) \wedge \min(X_{fst}, L) \wedge \max(X_{lst}, L) \wedge \\ & \mathit{adj3}(X_{fst}, X_0, X_1, L) \wedge \bigwedge_{V \in \mathcal{V}} V \in L \wedge \\ & \bigwedge_{\textit{for different } V, V' \in \mathcal{V}} V \neq V', \end{aligned} \tag{13}$$

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 (13) 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.

15 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 $s_1 \ldots s_n \in \{0,1\}^+$.

As a first approximation to represent the initial ID $\$q_0 s_1 \ldots s_n b \ldots b\$$ of M, let us select fresh different variables $X_2, \ldots, X_n$, 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):

$$\begin{array}{lll} IC(C^{t+2}) & \equiv_{df} & \langle X_{fst}, END\rangle \in C \wedge \langle X_0, Q_0\rangle \in C \wedge \langle X_{lst}, END\rangle \in C \wedge \\ & & \bigwedge_{i=1}^{n-1} succ(X_i, X_{i+1}, L) \wedge \bigwedge_{i=1}^{n} \langle X_i, S_i\rangle \in C \wedge \\ & & \forall u (X_n \subsetneq u \subsetneq X_{lst} \Rightarrow \langle u, BLANK\rangle \in C) \wedge \\ & & \forall u, v, w (\langle u, v\rangle \in C \wedge \langle u, w\rangle \in C \Rightarrow v = w). \end{array} \qquad (14)$$

The last two universal subformulas in (14) say that the input is padded with blanks and that C is a 'function', i.e., every tape symbol is *uniquely* defined.

We now can write the subformula $I(R)$ of (9):

$$I(R^{t+5}) \quad \equiv_{df} \quad \exists X^{t+2}, Y^{t+2} \Big(IC(X) \wedge \langle X, Y\rangle \in R \Big). \qquad (15)$$

Note that the type of $\langle X, Y\rangle$ in (15) is $t+4$ (see Section 5), hence the type of R is $t+5$. Recall that types in the atomic formulas of Ω should differ by one: $x^k \in y^{k+1}$.

Remark 15.1 The only drawback of the formula (14) (consequently, of (15)) 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 indices, 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 (14) is of length *linear in the size of input* (even written in binary), the formula (14) 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 (14) is (see (8) in Corollary 3.5)

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

(still, this is a faster than linearly growing stack of twos).

For the succinct minimally typed Ω we can get with the initial formula (14) a *stronger* lower bound of the form (see (7) of Corollary 3.5)

$$\exp_\infty(\exp(\frac{cn}{\log(n)})).$$

Below in Section 19 we describe a more economic way to represent an input. Nevertheless, the solution with the initial formula (14) 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 18.

16 Final ID, Subformula $F(R)$

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

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

We now can write the subformula $F(R)$ of (9):

$$F(R^{t+5}) \quad \equiv_{df} \quad \exists X^{t+2}, Y\Big(FC(X) \wedge \langle Y, X\rangle \in R\Big). \tag{16}$$

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

17 Correct Transitions, Subformula $C(R)$

The following lemma, now belonging to folklore, due to (Stockmeyer 1974, 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, j-th, 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. Formally,

Lemma 17.1 (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 properties:

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

$$\begin{array}{lllllllllll} \$\mathbf{d}_1\$ & = & d_{10} & d_{11} & \dots & d_{1j-1} & d_{1j} & d_{1j+1} & \dots & d_{1k} & d_{1k+1} \\ \$\mathbf{d}_2\$ & = & d_{20} & d_{21} & \dots & d_{2j-1} & d_{2j} & d_{2j+1} & \dots & d_{2k} & d_{2k+1} \end{array}$$

  one has $\mathbf{d_2} \in \mathit{Next}_M(\mathbf{d_1})$ if and only if

$$d_{2j-1}d_{2j}d_{2j+1} \in N_M(d_{1j-1}d_{1j}d_{1j+1})$$

  for all $j \in \{1, \dots, k\}$. $\square$

Note that the graph of any function N_M in Lemma 17.1 is of size *polynomial* in $\mathit{card}(\Delta)$ [7], by Remark 4.1 *does not depend on input*, and thus is *considered constant* (recall that we fix M before starting to construct a reduction to Ω).

This graph may be defined by the following boolean formula

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

$$\Rightarrow \bigvee_{s'_1s'_2s'_3 \in N_M(s_1s_2s_3)} (x' = s'_1 \wedge y' = s'_2 \wedge z' = s'_3) \Big).$$

Remark 17.2 The size of this formula is polynomial in $\mathit{card}(\Delta)$ (which is a 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. $\square$

We are now ready to write the formula $C(R)$ of (9):

$$\begin{array}{lll} C(R) \equiv_{\mathit{df}} \forall X, Y \Big(& \langle X, Y \rangle \in R^{t+5} \Rightarrow & \\ & \forall x, y, z, a, b, c, a', b', c' \Big[\mathit{adj3}(x, y, z, L) \wedge & \\ & \langle x, a \rangle \in X \wedge \langle y, b \rangle \in X \wedge \langle z, c \rangle \in X \wedge & \\ & \langle x, a' \rangle \in Y \wedge \langle y, b' \rangle \in Y \wedge \langle z, c' \rangle \in Y & (17) \\ & \Rightarrow \Psi_M(a, b, c, a', b', c') \Big] \wedge & \\ & \Big[\neg \exists x (\langle x, Q_a \rangle \in Y \Rightarrow \exists Z \langle Y, Z \rangle \in R \Big] \Big). & \end{array}$$

[7] This is because the largest subset of Δ^3 is of polynomial size and the graph specifies one such subset for each triple in Δ^3.

(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 (9) expressing the fact that a given $\exp_\infty(\exp(n)) - 2$-space bounded TM M accepts an input x.

18 Lower Bounds for Ω with Fixed Quantifier Prefix

The subformulas A, C, F of $\Phi_{M,x}$ (defined respectively by (13), (17), (16), (9)) are *quasi-fixed*, hence, *linearly bounded* in the length of input n. The initial subformula I of $\Phi_{M,x}$ defined by (15), (14) is of size:

- $O(n^2)$ for the verbose fully typed Ω,
- $O(n\log(n))$ 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 18.1 (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:)

$$\exp_\infty(\exp(\sqrt{c|\Phi|})) \tag{18}$$

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

(For partially typed Ω:)

$$\exp_\infty(\exp(c|\Phi|/\log(|\Phi|))). \tag{19}$$

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

Note that already (18), (19) provide a *superlinear 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 to represent an input. The solution with formula (14) (the only 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 (18) up to $\exp_\infty(\exp(c|\Phi|/\log(|\Phi|)))$ and (19) to $\exp_\infty(\exp(cn))$.

19 More Succinct Initial Formula

The non-quasi-fixed initial formula (15) – (14) in Section 15 was constructed by using $O(n)$ variables, where n is the length of input. This non-economic representation led to non-optimal lower bounds of Theorem 18.1. In this section we describe a cleverer way to represent an input by using only logarithmically many variables. We split the job into two subtasks. First, in Section 19.1, we describe a method to represent an input by a formula linear in the length of input. Second, in Sections 19.2 – 19.4, we describe how to 'copy' the input represented that way onto the initial ID of a TM.

19.1 Input Formula

Given a sequence of binary bits $b_j, \ldots, b_1$, let $br(b_j \ldots b_1)$ denote the natural number represented in binary by $b_j \ldots b_1$.

Let an input $s_1 \ldots s_n \in \{0,1\}^+$ of length n be given, and let $m = \lceil \log(n) \rceil$. We will show how to construct the formula (with variables d_i of type 0 and x of type 1)

$$INPUT_m(d_m, d_{m-1}, \ldots, d_2, d_1, x),$$

of size linearly bounded in n with the following property:

> When $d_m, d_{m-1}, \ldots, d_2, d_1$ are assigned binary values 0, 1, the formula $INPUT_m(d_m, d_{m-1}, \ldots, d_2, d_1, x)$ is true if and only if $x = s_k$, where $k = br(d_m d_{m-1} \ldots d_2 d_1) + 1$.

If necessary, we padd an input with blanks to the length 2^m; this explains why x is of type 1 — type 0 has only two elements, insufficient to represent the third value 'blank'.

Thus, the formula of size $O(n)$

$$\forall d_m, d_{m-1}, \ldots, d_2, d_1, x(INPUT_m(d_m, d_{m-1}, \ldots, d_2, d_1, x)$$

will say that the input is $s_1 \ldots s_n$ (padded with blanks to the length 2^m, whenever necessary).

To clarify the intuition and to simplify length counting, let us write explicitly the formula $INPUT_4(d_4, d_3, d_2, d_1, x)$, as an example.

First observe that we cannot straightforwardly write:

$$\text{16 lines} \begin{cases} (d_4 = 0 \;\wedge\; d_3 = 0 \;\wedge\; d_2 = 0 \;\wedge\; d_1 = 0 \;\Rightarrow\; x = s_1) \quad \wedge \\ \qquad\qquad \ldots \\ (d_4 = 1 \;\wedge\; d_3 = 1 \;\wedge\; d_2 = 1 \;\wedge\; d_1 = 1 \;\Rightarrow\; x = s_{16}), \end{cases}$$

because in this case each variable appears n times and there are $O(\log(n))$ different variables. Thus the formula grows at least as $O(n\log(\log(n)))$, faster than we can afford.

Here is the correct way to go, if we need a formula of size $O(n)$:
$INPUT_4(d_4, d_3, d_2, d_1, x) \ \equiv_{df}$

$$\begin{array}{llllllllllll}
(& d_4 = 0 & \Rightarrow & (& d_3 = 0 & \Rightarrow & (& d_2 = 0 & \Rightarrow & (& d_1 = 0 \Rightarrow & x = s_1 \quad) & \wedge \\
 & & & & & & & & & (& d_1 = 1 \Rightarrow & x = s_2 \quad)) & \wedge \\
 & & & & & & (& d_2 = 1 & \Rightarrow & (& d_1 = 0 \Rightarrow & x = s_3 \quad) & \wedge \\
 & & & & & & & & & (& d_1 = 1 \Rightarrow & x = s_4 \quad))) & \wedge \\
 & & & (& d_3 = 1 & \Rightarrow & (& d_2 = 0 & \Rightarrow & (& d_1 = 0 \Rightarrow & x = s_5 \quad) & \wedge \\
 & & & & & & & & & (& d_1 = 1 \Rightarrow & x = s_6 \quad)) & \wedge \\
 & & & & & & (& d_2 = 1 & \Rightarrow & (& d_1 = 0 \Rightarrow & x = s_7 \quad) & \wedge \\
 & & & & & & & & & (& d_1 = 1 \Rightarrow & x = s_8 \quad)))) & \wedge \\
(& d_4 = 1 & \Rightarrow & (& d_3 = 0 & \Rightarrow & (& d_2 = 0 & \Rightarrow & (& d_1 = 0 \Rightarrow & x = s_9 \quad) & \wedge \\
 & & & & & & & & & (& d_1 = 1 \Rightarrow & x = s_{10} \quad)) & \wedge \\
 & & & & & & (& d_2 = 1 & \Rightarrow & (& d_1 = 0 \Rightarrow & x = s_{11} \quad) & \wedge \\
 & & & & & & & & & (& d_1 = 1 \Rightarrow & x = s_{12} \quad))) & \wedge \\
 & & & (& d_3 = 1 & \Rightarrow & (& d_2 = 0 & \Rightarrow & (& d_1 = 0 \Rightarrow & x = s_{13} \quad) & \wedge \\
 & & & & & & & & & (& d_1 = 1 \Rightarrow & x = s_{14} \quad)) & \wedge \\
 & & & & & & (& d_2 = 1 & \Rightarrow & (& d_1 = 0 \Rightarrow & x = s_{15} \quad) & \wedge \\
 & & & & & & & & & (& d_1 = 1 \Rightarrow & x = s_{16} \quad)))). &
\end{array}$$

Here the variable x occurs 2^4 times, d_1 occurs 2^4 times, d_2 occurs 2^3 times, d_3 occurs 2^2 times, d_4 occurs 2^1 times. This count will be useful in the sequel and will show that $INPUT_m(d_m, d_{m-1}, \ldots, d_2, d_1, x)$ is *linearly bounded* in $n = 2^m$.

It is clear how to spell out the analogous (to $m = 4$) 'case analysis' formula $INPUT_m(d_m, d_{m-1}, \ldots, d_2, d_1, x)$ for every $m = \lceil \log(n) \rceil$ and sequence of input bits $s_1 \ldots s_n$. Define by induction on $i = 0, \ldots, m$ the formulas $input_{i,j}(d_i, \ldots, d_1)$, where $j = 0, \ldots, 2^{m-i} - 1$, by:

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

It remains to define

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

Even if we write the indices of variables in unary, the total space occupied by indices of variables d_i's in $INPUT_m(d_m, d_{m-1}, \dots, d_2, d_1, x)$ will be equal to (see the discussion after the example above)

$$\sum_{k=1}^{m} k \cdot 2^{m-k+1},$$

which may be easily shown, by induction (or by using Maple), equal to

$$4 \cdot 2^m - 2m - 2 < 4 \cdot 2^m = O(n).$$

The formula $INPUT_m(d_m, d_{m-1}, \dots, d_2, d_1, x)$ additionally contains the linear number of occurrences of x (indexed with index 0, which occupies no extra space written in unary), 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 *linearly bounded* by the length of input n, as needed, both in succinct and fully typed versions of Ω. Clearly, the formula $INPUT_m$ can be constructed in polynomial time.

19.2 Counting Long Distances in a Chain

Thus, by using the formula $INPUT_m(d_m, d_{m-1}, \dots, d_2, d_1, x)$ of size $O(n)$ we can express any input sequence of bits $s_1 \dots s_n$ of length n.

Recall that to write the initial formula (14), (15) we have to say, for any input sequence of bits $s_1 \dots s_n$, 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 $s_i = 1$. Therefore, it remains to 'copy' the input $s_1 \dots s_n$, represented by the formula $INPUT_m$ on the initial tape. In Section 15, Remark 15.1, we discussed already that the straightforward method fails, because it needs n new variables of type $t-1$ (of size $O(n)$). Only writing the the indices (in binary) of these n variables requires space $O(n \log(n))$, and thus cannot lead to a linearly bounded reduction. Even worse, since in the verbose version of Ω we have to annotate each occurrence of a variable with its type, this leads to at least quadratic reduction with growth rate $O(n^2)$.

To be able to address n successors in a chain L more economically, we will define the formulas

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

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 (12)), provided $n_2 \geq n_1$.

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 $2^m - 1$-th) 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 $m = \lceil \log(n) \rceil$) succeeding elements in L^t starting with X_1 index values $s_1, \dots, s_{2^m}$ as follows:

$$
\begin{array}{ll}
D(X_1) \equiv_{df} \forall d_m, \dots, d_1, v, Y, V \Big(& \\
\quad INPUT_m(d_m, \dots, d_1, v) & \wedge \\
\quad SUCC_m(X_1, \underbrace{0, \dots, 0}_{m};\ Y, d_m, \dots, d_1) & \Rightarrow \\
\Rightarrow \quad ((v = \{1\} \Leftrightarrow V = ONE) \wedge (v = \{0\} \Leftrightarrow V = ZERO) & \wedge \\
\quad (v = \{0,1\} \Leftrightarrow V = BLANK) \wedge \langle Y, V \rangle \in C \Big), &
\end{array}
$$

where Y, V are of type $t-1$, v of type 1, and all other variables of type 0. We use the variable v of type 1 to represent three possibilities in the input: $\{1\}$ for 1, $\{0\}$ for 0, and $\{0,1\}$ for the blank (recall that we padd inputs to length $2^{\lceil \log(n) \rceil}$ with blanks).

Henceforth, the subformula IC of the initial formula (15) may be defined more economically (as compared with (14)) as follows:

$$
\begin{array}{lll}
IC(C^{t+2}) & \equiv_{df} & \langle X_{fst}, END \rangle \in C \wedge \langle X_0, Q_0 \rangle \in C \wedge \\
& & \langle X_{lst}, END \rangle \in C \wedge D(X_1) \wedge \\
& & \forall Y Z (SUCC_m(X_1, 0, \dots, 0;\ Y, 1, \dots, 1) \wedge \\
& & \qquad Y \subsetneq Z \subsetneq X_{lst} \Rightarrow \langle Z, BLANK \rangle \in C) \wedge \\
& & \forall u, v, w (\langle u, v \rangle \in C \wedge \langle u, w \rangle \in C \Rightarrow v = w).
\end{array} \tag{20}
$$

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

$$
\begin{array}{lllll}
x^0 & < & y^0 & \equiv_{df} & x \in \{0\} \wedge y \in \{1\}, \\
x^0 & \leq & y^0 & \equiv_{df} & \neg y < x.
\end{array}
$$

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 define:

$$
\begin{aligned}
&SUCC_0(X^{t-1}, d_1;\ Y^{t-1}, e_1) \equiv_{df} \\
&\qquad (\neg e_1 < d_1) && \wedge \\
&\qquad (d_1 = e_1 \quad \Rightarrow \quad X = Y) && \wedge \\
&\qquad (d_1 < e_1 \quad \Rightarrow \quad succ(X, Y, L^t)). &&
\end{aligned} \tag{21}
$$

For $i \geq 0$ define, inductively:

$$
\begin{aligned}
&SUCC_{i+1}(X^{t-1}, d_{i+1}, d_i, \dots, d_1;\ Y^{t-1}, e_{i+1}, e_i, \dots, e_1) \equiv_{df} \\
&\quad (\neg e_{i+1} < d_{i+1}) && \wedge \\
&\quad (d_{i+1} = e_{i+1} \quad \Rightarrow \quad SUCC_i(X, d_i, \dots, d_1;\ Y, e_i, \dots, e_1)) && \wedge \\
&\quad \Big(d_{i+1} < e_{i+1} \quad \Rightarrow \quad \exists Z_1^{t-1}, Z_2 \Big[\\
&\qquad\qquad SUCC_i(X, d_i, \dots, d_1;\ Z_1, \underbrace{1, \dots, 1}_{i}) && \wedge \\
&\qquad\qquad SUCC_i(Z_2, \underbrace{0, \dots, 0}_{i};\ Y, e_i, \dots, e_1) && \wedge \\
&\qquad\qquad succ(Z_1, Z_2, L^t)\Big]\Big).
\end{aligned} \tag{22}
$$

It is easy to see (by induction) that $SUCC_m$ defined by (21), (22) allows us to count distances up to 2^m between elements of the chain L^t.

The drawback of the definition (22) is that

$$SUCC_m(X^{t-1}, d_m, \dots, d_1;\ Y^{t-1}, e_m, \dots, e_1),$$

fully expanded by using (21) and (22) to a formula containing no occurrences of $SUCC$, will contain $O(2^m) = O(n)$ occurrences of variables X, Y. This is easy to see: if $SUCC_i(X, \dots)$ contains k occurrences of X after full expansion, then $SUCC_{i+1}(X, \dots)$ will contain $2k$ such occurrences. Thus, we do not gain anything with definition (22), as compared with the straightforward method with n new variables described in Section 15. However, we can do much better, as shown in the next section.

19.3 Abbreviation Trick

The right-hand side of (22) 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; cf., (Ferrante & Rackoff 1979, Compton & Henson 1990). Here we simplify this trick for our needs and adapt it for the case when all multiple subformula occurrences are *positive*. As an advantage we do not need the equivalence connective $\Leftrightarrow$. This section allows us to keep the paper self-contained and helps in counting formula sizes in the sequel.

Lemma 19.1 Given a quantifier-free formula Φ containing m positive occurrences

$$P(\bar{t}_i), \text{ for } i = 1, \ldots, m$$

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 *linearly bounded* by the size of Φ, containing just one positive occurrence of P and no negative occurrences of P. More precisely, the formula Δ is:

$$\begin{aligned}\exists x_1 y_1 \ldots x_m y_m \Big(\Phi' \wedge \forall uv\bar{z}\Big[\; & \textstyle\bigvee_{i=1}^{m}(u = x_i \wedge v = y_i \wedge \bar{z} = \bar{t}_i) \Rightarrow \\ & \Rightarrow (u = v \Rightarrow P(\bar{z}))\Big]\Big),\end{aligned} \tag{23}$$

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

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

Proof. 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 i-th occurrence of $P(\bar{t}_i)$ with equality $x_i = y_i$. . Note that $m = O(n/\log(n))$, 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 $O(\log(n))$ bits.

Let us show that Φ is equivalent to

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

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 are true:

$$\begin{aligned} &\Phi' \equiv \Phi\Big[x_i = y_i / P(\bar{t}_i)\Big]_{i=1}^{m}, \\ &\bigwedge_{i=1}^{m}(x_i = y_i \Rightarrow P(\bar{t}_i)). \end{aligned} \tag{24}$$

Recall that Φ is positive in $P(\bar{t}_i)$, hence, by construction, Φ' is positive in $x_i = y_i$. Therefore, Φ' is monotone in $x_i = y_i$. This and (24) imply that $\Phi'\Big[P(\bar{t}_i)/x_i = y_i\Big]_{i=1}^{m}$ 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

$$\psi \equiv \bigwedge_{i=1}^{m}(x_i = y_i \Rightarrow P(\bar{t}_i))$$

of Ψ containing m occurrences of P is equivalent to

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

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. $\square$

19.4 Complexity of SUCC

Applying Lemma 19.1, and putting all quantifiers in front of the formula, we can rewrite the definition (22) of $SUCC_{i+1}$ equivalently by using just one occurrence of $SUCC_i$ as follows:

$SUCC_{i+1}(X^{t-1}, d_{i+1}, d_i, \ldots, d_1;\ Y^{t-1}, e_{i+1}, e_i, \ldots, e_1) \ \equiv_{df}$

$$\exists Z_1, Z_2, x_1, y_1, x_2, y_2, x_3, y_3 \; \forall u, v, Z, z_1, \ldots, z_i, Z', z'_1, \ldots, z'_i \; \Big\{$$

$$\begin{aligned}
&(\neg e_{i+1} < d_{i+1}) \quad \wedge \\
&(d_{i+1} = e_{i+1} \quad \Rightarrow \quad x_1 = y_1) \quad \wedge \\
&(d_{i+1} < e_{i+1} \quad \Rightarrow \quad x_2 = y_2 \quad \wedge \quad x_3 = y_3 \quad \wedge \quad succ(Z_1, Z_2, L^t))\wedge
\end{aligned} \tag{25}$$

$$\begin{aligned}
\wedge \Big[&(u = x_1 \wedge v = y_1 \wedge Z = X \wedge Z' = Y \wedge \textstyle\bigwedge_{j=1}^{i}(z_j = d_j \wedge z'_j = e_j))\vee \\
&(u = x_2 \wedge v = y_2 \wedge Z = X \wedge Z' = Z_1 \wedge \textstyle\bigwedge_{j=1}^{i}(z_j = d_j \wedge z'_j = 1))\vee \\
&(u = x_3 \wedge v = y_3 \wedge Z = Z_2 \wedge Z' = Y \wedge \textstyle\bigwedge_{j=1}^{i}(z_j = 0 \wedge z'_j = e_j)) \quad \Rightarrow
\end{aligned}$$

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

Therefore, each iteration expanding $SUCC_{i+1}$ via $SUCC_i$ using the definition (25) introduces the following new quantified variables:

1. four variables of type $t-1$, namely Z_1, Z_2, Z, Z',
2. eight variables of type 0, namely, $x_1, y_1, x_2, y_2, x_3, y_3, u, v$,
3. $2i$ variables of type 0, namely, $z_1, \ldots, z_i, z'_1, \ldots, z'_i$.

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,
3. $O(m)$ new quantified variables of type 0, which is superseded by
4. $O(m^2)$ variables of type 0.

(We could do even better by reusing variable names. Thus the total number of different quantified variables would be *fixed*. However this is not needed for our purposes, and does not lead to any further improvement.)

By definition (25) of $SUCC$, the full expansion of $SUCC_m$, into a formula without occurrences of $SUCC$, will contain:

1. $2m - 1 = O(m) = O(\log(n))$ quantifier alternations,

2. $O(m)$ occurrences of variables of type $t-1$; in fact, each such variable occurs only a *fixed* number of times (see (25), where each of X, Y appears twice Z_1, Z_2 appears three times (quantified, in *succ*, in equalities), and Z, Z' will appear six times each (quantified, three times in equalities shown, and, by induction, twice in $SUCC_i$)),

3. similarly, a fixed number of occurrences of each of the $O(m^2)$ variables of type 0,

4. similarly, $O(m^2)$ occurrences of boolean connectives and parentheses.

Recall that $m = \lceil \log(n) \rceil$, thus, 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 indices to represent k different variables). This is smaller than $O(n)$, and thus leads to a linearly bounded initial formula (20) 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 type $t-1$ should be annotated with type $t-1$, of length $O(n)$. Therefore, the formulas $SUCC_m$ and (20) are of superlinear length $O(n \log(n))$ in the case of verbose fully typed Ω.

Note that 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.

Remark 19.2 Note that this superlinear explosion does not occur in the first-order theories, which do not require variable type annotations. Compare:

1. For the first-order theories, introducing logarithmically many new variables or having logarithmically many variable occurrences is of no danger, and does not lead to a superlinear formula growth (the same is true for 'logarithmically' replaced with 'polylogarithmically').

2. For theories with typed variables, as verbose Ω introducing more than a constant number of new variables, or having more than a constant number of variable occurrences may lead to *superlinear* explosion, when types of variables linearly depend on the size of input. $\square$

Remark 19.3 The above $O(n \log(n))$ *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 *sublinear*, and the reduction taking an input of length n into the formula $SUCC_m$, with $m = \lceil \log(n) \rceil$ is *linearly bounded.* $\square$

20 Stronger Lower Bounds

The initial formula $I(R)$ of (9) defined by (14), (15) was the only non-quasi-fixed and non-linearly bounded formula in the construction preceding Section 18. In Section 19 we constructed a more succinct initial formula $I(R)$ of size

- $O(n \log(n))$ in the case of fully typed Ω,
- $O(n)$ in the case of partially typed Ω.

Therefore, Lemma 3.3 and Corollary 3.5 apply, and we get the following

Theorem 20.1 (Lower Bounds for Ω) Every decision algorithm for Ω requires space exceeding, respectively,

(For Verbose, Fully typed Ω:) $\exp_\infty(\exp(\dfrac{c \cdot |\Phi|}{\log(|\Phi|)}))$ for some constant $c > 0$ and infinitely many sentences Φ of verbose Ω.

(For Succinct, Minimally typed Ω:) $\exp_\infty(\exp(c \cdot |\Phi|))$ for some constant $c > 0$ and infinitely many sentences Φ of succinct Ω. $\square$

This finishes the proof of the Main Theorem.

21 Concluding Remarks

Remark 21.1 (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 19 to write a more succinct initial formula representing an input. This is a big advantage, because otherwise:

- 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 poorer lower bounds.

- We would fail to have the fixed quantifier prefix complexity results of Theorem 18.1. The formula $\Phi_{M,x}$ in (9) we construct before Section 18 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 (Kuper & Vardi 1993), which needs more and more quantifier alternations to get increase in complexity. $\square$

Remark 21.2 (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 (Vorobyov 1997) based on the uniform lower bound method due to (Compton & Henson 1990), 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 (Compton & Henson 1990) spend extra effort on proving stronger *inseparability* results, which imply lower bounds that hold not only to theories, but to all their subtheories. $\square$

Remark 21.3 (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. $\square$

Remark 21.4 (Any 'More Nonelementary' Theories?) The following challenging problem in (Compton & Henson 1990, Problem 10.12) is open/closed (modulo what is considered 'natural'):

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

In (Vorobyov 1997) 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). $\Box$

Remark 21.5 (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 $O(n(\log(n))$ non-linearly bounded formula we used was *SUCC* in Section 19.4 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?

References

Compton, K. J. & Henson, C. W. (1990), 'A uniform method for proving lower bounds on the computational complexity of logical theories', *Annals Pure Appl. Logic* **48**, 1–79.

Ferrante, J. & Rackoff, C. W. (1979), *The computational complexity of logical theories*, Vol. 718 of *Lect. Notes Math.*, Springer-Verlag.

Henkin, L. (1963), 'A theory of propositional types', *Fundamenta Mathematicæ* **52**, 323–344.

Hopcroft, J. E., & Ullman, J. D. (1979), *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Company.

Hull, R. & Su, J. (1991), 'On the expressive power of database queries with intermediate types', *J. Comput. Syst. Sci.* **43**, 219–267.

Kuper, G. & Vardi, M. (1993), 'On the complexity of queries in the logical data model', *Theor. Comput. Sci.* **116**, 33–57.

Lewis, H. R. (1980), 'Complexity results for classes of quantificational formulas', *J. Comput. Syst. Sci.* **21**, 317–353.

Mairson, H. (1992), 'A simple proof of a theorem of Statman', *Theor. Comput. Sci.* **103**, 387–394.

Meyer, A. R. (1974), The inherent computational complexity of theories of ordered sets, *in* 'International Congress of Mathematicians', Vancouver, pp. 477–482.

Meyer, A. R. (1975), 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, pp. 132–154.

Statman, R. (1979), 'The typed λ-calculus is not elementary recursive', *Theor. Comput. Sci.* **9**, 73–81.

Stockmeyer, L. J. (1974), The complexity of decision problems in automata theory and logic, PhD thesis, MIT Lab for Computer Science. (Also /MIT/LCS Tech Rep 133).

Stockmeyer, L. J. (1977), 'The polynomial-time hierarchy', *Theor. Comput. Sci.* **3**, 1–22.

Stockmeyer, L. J. (1987), 'Classifying the computational complexity of problems', *J. Symb. Logic* **52**(1), 1–43.

Stockmeyer, L. J. & Meyer, A. R. (1973), Word problems requiring exponential time: preliminary report, *in* '5th Symp. on Theory of Computing', pp. 1–9.

Vorobyov, S. (1997), The "hardest" natural decidable theory, *in* G. Winskel, ed., '12th Annual IEEE Symp. on Logic in Computer Science (LICS'97)', IEEE, pp. 294–305. Available from `http://www.mpi-sb.mpg.de/~sv`.

Vorobyov, S. & Voronkov, A. (1998), 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)', p. ??? (to appear). Preliminary full version in: Research Report MPI-I-97-2-010, Max-Planck Institut für Informatik, Saarbrücken, November 1997. Available from `http://data.mpi-sb.mpg.de/internet/reports.nsf/NumberView/97-2-010`.