Theory Comput Syst (2014) 55:421–447
DOI 10.1007/s00224-013-9465-0

Two-Way Automata Versus Logarithmic Space

Christos A. Kapoutsis

Published online: 11 April 2013

Abstract We strengthen a previously known connection between the size complexity of two-way finite automata (2FAs) and the space complexity of Turing machines (TMs). Specifically, we prove that

– every s-state 2NFA has a poly(s)-state 2DFA that agrees with it on all inputs of length $\leq s$ if and only if $\mathsf{NL} \subseteq \mathsf{L/poly}$, and
– every s-state 2NFA has a poly(s)-state 2DFA that agrees with it on all inputs of length $\leq 2^s$ if and only if $\mathsf{NLL} \subseteq \mathsf{LL/polylog}$.

Here, 2DFAs and 2NFAs are the deterministic and nondeterministic 2FAs, NL and L/poly are the standard classes of languages recognizable in logarithmic space by nondeterministic TMs and by deterministic TMs with access to polynomially long advice, and NLL and LL/polylog are the corresponding complexity classes for space $O(\log\log n)$ and advice length poly($\log n$). Our arguments strengthen and extend an old theorem by Berman and Lingas and can be used to obtain variants of the above statements for other modes of computation or other combinations of bounds for the input length, the space usage, and the length of advice.

Keywords Two-way finite automata · 2D versus 2N · Sakoda-Sipser conjecture · Logarithmic space · L versus NL · Sub-logarithmic space

Preliminary version presented in the 6th International Computer Science Symposium in Russia, St. Petersburg, 14–18 June 2011 [Lecture Notes in Computer Science vol. 6651, Springer-Verlag, pp. 359–372].

Research funded by a Marie Curie Intra-European Fellowship (PIEF-GA-2009-253368) within the European Union Seventh Framework Programme (FP7/2007-2013).

C.A. Kapoutsis (✉)
LIAFA, Université Paris Diderot—Paris VII, Case 7014, 75205 Paris Cedex 13, France
e-mail: christos.kapoutsis@liafa.jussieu.fr

1 Introduction

The question whether nondeterministic computations can be more powerful than deterministic ones is central in complexity theory. Numerous instantiations have been studied, for a variety of computational models under a variety of resource restrictions. This article investigates the relationship between two families of such instantiations, those for *two-way finite automata* under *size* restrictions and those for *Turing machines* under *space* restrictions. We start by introducing each of these two families separately, then continue to describe the relationship between them.

1.1 Size-Bounded Two-Way Finite Automata

The question whether nondeterminism makes a difference in two-way finite automata (2FAs) of bounded size was first studied by Seiferas in the early 70s [15]. Formally, this is the question whether every nondeterministic 2FA (2NFA) has an equivalent deterministic 2FA (2DFA) with only polynomially more states:

$$\begin{array}{l}\text{Is there a polynomial } p \text{ such that every } s\text{-state 2NFA has}\\ \text{a 2DFA with } \leq p(s) \text{ states that agrees with it on all possible inputs?}\end{array} \tag{1}$$

The answer has long been conjectured to be negative [13, 15]. Indeed, this has been confirmed in several special cases: when the automata are *single-pass* (i.e., they halt upon reaching an end-marker [15]), or *almost oblivious* (i.e., they exhibit $o(n)$ distinct input head trajectories over all n-long inputs [7]), or *moles* (i.e., they explore the configuration graph implied by the input [9]), or they perform *few reversals* (i.e., they reverse their input head only $o(n)$ times on every n-long input [10]). For *unary* automata, however, a non-trivial upper bound is known: an equivalent 2DFA with only quasi-polynomially more states always exists [5].

A robust theoretical framework was built around this question by Sakoda and Sipser in the late 70s [13]. Central in this framework are the complexity classes 2D and 2N. The former consists of every family of languages $(L_h)_{h\geq 1}$ that can be recognized by a polynomial-size family $(A_h)_{h\geq 1}$ of 2DFAs, in the sense that every A_h recognizes the corresponding L_h and has $\leq p(h)$ states, for some polynomial p. The latter is the analogous class for 2NFAs. In these terms, the question becomes equivalent to the question:

$$\text{Is it true that } \mathsf{2D} \supseteq \mathsf{2N}? \tag{1'}$$

and the long-standing conjecture becomes equivalent to $\mathsf{2D} \not\supseteq \mathsf{2N}$.[1]

In this article, we consider 'bounded' variants of (1). We replace the requirement that the 2DFA and 2NFA agree on *all possible inputs* with the requirement that they

[1] Note the unusual forms '$\mathsf{A} \supseteq \mathsf{B}$' and '$\mathsf{A} \not\supseteq \mathsf{B}$'. In cases where $\mathsf{A} \subseteq \mathsf{B}$ (e.g., when $\mathsf{A} = \mathsf{2D}$ & $\mathsf{B} = \mathsf{2N}$), these are of course equivalent to the more usual '$\mathsf{A} = \mathsf{B}$' and '$\mathsf{A} \subsetneq \mathsf{B}$'. However, we will encounter cases where $\mathsf{A} \subseteq \mathsf{B}$ is not true (e.g., when $\mathsf{A} = \mathsf{2D}$ & $\mathsf{B} = \mathsf{2N/poly}$ or when $\mathsf{A} = \mathsf{L/poly}$ & $\mathsf{B} = \mathsf{NL}$—see below) and hence '$\mathsf{A} \supseteq \mathsf{B}$' and '$\mathsf{A} \not\supseteq \mathsf{B}$' are appropriate. We thus use these forms throughout the article, so that all statements are easy to compare. We read and think of these forms as 'A covers/does not cover B'.

agree only on *all 'short' inputs*, for different interpretations of 'short'. Could it be that under this relaxed requirement the 2DFA can now stay only polynomially larger?

For the most part, we will focus only on two interpretations of 'short'. Under the first one, an input is 'short' if its length is at most *exponential* in the size of the 2NFA:

$$\text{Is there a polynomial } p \text{ such that every } s\text{-state 2NFA has a 2DFA with } \leq p(s) \text{ states that agrees with it on all inputs of length } \leq 2^s? \tag{2}$$

Under the second interpretation, an input is 'short' if its length is at most *linear* in the size of the 2NFA:

$$\text{Is there a polynomial } p \text{ such that every } s\text{-state 2NFA has a 2DFA with } \leq p(s) \text{ states that agrees with it on all inputs of length } \leq s? \tag{3}$$

We conjecture that neither relaxation makes the simulation easy. Namely, much like the answer to (1), we conjecture that the answers to (2) and (3) are still negative.

The new questions can be expressed in the Sakoda-Sipser framework, as well. To this end, we first extend the definitions of 2D and 2N so that they now consist of all families of *promise problems* (as opposed to just all families of *languages*) that admit polynomial-size families of 2DFAs and 2NFAs, respectively. Then, we introduce two new complexity classes, subclasses of the extended 2N, called $\mathsf{2N/exp}$ and $\mathsf{2N/poly}$. The former consists of only those promise-problem families $(\mathfrak{L}_h)_{h\geq 1}$ in 2N where all instances of $\mathfrak{L}_h$ are of length $\leq 2^{q(h)}$, for some polynomial q and all h. Then (2) can be proved equivalent to:

$$\text{Is it true that } \mathsf{2D} \supseteq \mathsf{2N/exp}? \tag{2'}$$

Similarly, $\mathsf{2N/poly}$ consists of only those promise-problem families in 2N where every $\mathfrak{L}_h$ has instances of length $\leq q(h)$. Then (3) can be proved equivalent to:

$$\text{Is it true that } \mathsf{2D} \supseteq \mathsf{2N/poly}? \tag{3'}$$

This way, our stronger conjectures are that even $\mathsf{2D} \not\supseteq \mathsf{2N/exp}$ and $\mathsf{2D} \not\supseteq \mathsf{2N/poly}$.

We remark that other interpretations of 'short' are also possible, but not as crucial. In Sect. 2.1 (Lemma 2.1) we will see that restricting (1) to super-exponential lengths produces a question which is actually still equivalent to (1). Variants for quasi-polynomial lengths will be considered in Sect. 3 (Theorem 3.2).

1.2 Space-Bounded Turing Machines

The question whether nondeterminism makes a difference in Turing machines (TMs) of bounded space dates back to the work of Kuroda in the mid 60s [12]. Formally, this is the question whether every nondeterministic TM (NTM) has an equivalent deterministic TM (DTM) that uses the same space:

$$\text{Does every bound } f \text{ satisfy } \mathsf{DSPACE}\big(f(n)\big) \supseteq \mathsf{NSPACE}\big(f(n)\big)? \tag{4}$$

The answer has long been conjectured to be negative. We know, however, that we may restrict our attention to bounds outside $o(\log\log n)$, because for $f(n) = o(\log\log n)$

the inclusion follows trivially from the fact that both complexity classes contain only regular languages [1, 6, 17]. We also know that, if the inclusion is satisfied by any bound in $\Theta(\log\log n) \cup \Omega(\log n)$, then all greater bounds satisfy it, too [14, 18].

For the most part, we will focus on the two smallest natural bounds, $\log\log n$ and $\log n$. The former gives rise to the complexity classes $\mathsf{LL} := \mathsf{DSPACE}(\log\log n)$ and $\mathsf{NLL} := \mathsf{NSPACE}(\log\log n)$, and to the respective question:

$$\text{Is it true that } \mathsf{LL} \supseteq \mathsf{NLL}? \tag{5}$$

Similarly, the latter bound gives rise to the standard space complexity classes $\mathsf{L} = \mathsf{DSPACE}(\log n)$ and $\mathsf{NL} = \mathsf{NSPACE}(\log n)$, and to the question:

$$\text{Is it true that } \mathsf{L} \supseteq \mathsf{NL}? \tag{6}$$

Once again, the answers to both questions are conjectured to be negative; i.e., most people believe that both $\mathsf{LL} \not\supseteq \mathsf{NLL}$ and $\mathsf{L} \not\supseteq \mathsf{NL}$.

An early observation on (6) was that, even if $\mathsf{L} \not\supseteq \mathsf{NL}$, a deterministic logarithmic-space TM might still manage to simulate a nondeterministic one if it is allowed non-uniform behavior [2]. This led to the introduction of the class $\mathsf{L/poly}$ of all languages that can be recognized in space $O(\log n)$ by a DTM which has access to $\text{poly}(n) \equiv n^{O(1)}$ bits of non-uniform advice [11]. The new question was:

$$\text{Is it true that } \mathsf{L/poly} \supseteq \mathsf{NL}? \tag{6$^+$}$$

and the stronger conjecture was that even $\mathsf{L/poly} \not\supseteq \mathsf{NL}$. By a similar reasoning, we can introduce a corresponding class $\mathsf{LL/polylog}$ for space $O(\log\log n)$ and for $\text{poly}(\log n)$ bits of advice, then ask the question:

$$\text{Is it true that } \mathsf{LL/polylog} \supseteq \mathsf{NLL}? \tag{5$^+$}$$

and conjecture that even $\mathsf{LL/polylog} \not\supseteq \mathsf{NLL}$.

We remark that other combinations of bounds for the space and the length of advice are also possible, but not as crucial. For example, if both the space and advice length are $O(\log n)$, then the resulting question whether $\mathsf{L/log} \supseteq \mathsf{NL}$ is equivalent to (6) [11]. Combinations of sub-logarithmic space and sub-linear advice length will be considered in Sect. 3. Finally, we note that the classes $\mathsf{DSPACE}(f)/2^{O(f)}$ for varying f have been studied in [8] under the names $\mathsf{NUDSPACE}(f)$.

1.3 Size-bounded 2FAs Versus Space-Bounded TMs

It has long been known, by a theorem of Berman and Lingas [3], that a simulation of nondeterministic logarithmic-space TMs by deterministic ones would imply the simulation of nondeterministic polynomial-size 2FAs by deterministic ones on polynomially-long inputs, in our notation:

$$\mathsf{2D} \supseteq \mathsf{2N/poly} \impliedby \mathsf{L} \supseteq \mathsf{NL}.$$

Our contribution is the tightening and deepening of this relationship.

We first show that $\mathsf{2D} \supseteq \mathsf{2N/poly}$ follows even from the weaker assumption that $\mathsf{L/poly} \supseteq \mathsf{NL}$, and that then the converse is also true:

Theorem 1.1 $\mathsf{2D} \supseteq \mathsf{2N/poly} \iff \mathsf{L/poly} \supseteq \mathsf{NL}$.

Next, we prove a similar relationship for the case where the bound for the input lengths increases to exponential while the bounds for the space and the advice length decrease to log-logarithmic and poly-logarithmic, respectively.

Theorem 1.3 $\mathsf{2D} \supseteq \mathsf{2N/exp} \iff \mathsf{LL/polylog} \supseteq \mathsf{NLL}$.

Finally, in between Theorems 1.1 and 1.3, we also relate the different levels of quasi-polynomial input lengths with different levels of sub-logarithmic space and sub-linear lengths of advice:

Theorem 1.2 *For all $k \geq 1$ we have*:

$$\mathsf{2D} \supseteq \mathsf{2N}/2^{O(\log^k h)} \iff \mathsf{DSPACE}\bigl(\sqrt[k]{\log n}\bigr)/2^{O(\sqrt[k]{\log n})} \supseteq \mathsf{NSPACE}\bigl(\sqrt[k]{\log n}\bigr).$$

(See Sect. 2 for the precise definitions of the notations used in this statement.)

For the most part, the proofs of these theorems elaborate on standard, old ideas [3, 13, 18]. Perhaps their main value lies in what they imply for how we approach the questions on the two sides of the equivalences, as the next two paragraphs explain.

On the one hand, people interested in size-bounded 2FAs can use these theorems to extract evidence about how hard it is for various proof strategies towards $\mathsf{2D} \not\supseteq \mathsf{2N}$ to succeed. For example, suppose that we are exploring two strategies, the first of which promises to eventually establish that every deterministic simulation of 2NFAs fails on at most exponentially long inputs, while the second strategy promises to prove failure only on super-exponentially long inputs. Then the first strategy is probably harder to succeed, because it implies an additional breakthrough in understanding nondeterminism in space-bounded TMs (by Theorem 1.3), while the second strategy involves no such implications. (Incidentally, all proof techniques that have been successful on restricted 2FAs are of the latter kind.)

On the other hand, people interested in space-bounded TMs can find in the 2D versus 2N question a single unifying setting in which to work. Separating polynomial-size 2NFAs from polynomial-size 2DFAs on super-exponentially long inputs can be seen as the first step in a gradual approach that sees NTMs being separated from DTMs for larger and larger space bounds as improved proof techniques separate polynomial-size 2NFAs from polynomial-size 2DFAs on shorter and shorter inputs.

Another aspect of the proofs of Theorems 1.1, 1.2, and 1.3 is that they can be modified so as to work also for other modes of computation. For example, consider *alternating* 2FAs and TMs, and let 2A/poly and AL denote the alternating analogues of the complexity classes 2N/poly and NL, respectively. As with nondeterminism, the inclusions $\mathsf{2D} \supseteq \mathsf{2A/poly}$ and $\mathsf{L} \supseteq \mathsf{AL}$ are open and conjectured to be false. Recalling that alternating logarithmic space equals deterministic polynomial time, namely

$$\mathsf{AL} = \mathsf{P} = \mathsf{DTIME}\bigl(\mathrm{poly}(n)\bigr),$$

we can state the following analogue of Theorem 1.1:

Theorem 1.4 $\mathsf{2D} \supseteq \mathsf{2A/poly} \iff \mathsf{L/poly} \supseteq \mathsf{P}$.

We claim that its proof is a straightforward modification of that of Theorem 1.1.

Concluding this introduction, we mention a different strengthening of the theorem of Berman and Lingas, by Geffert and Pighizzini [4]: if $\mathsf{2N/unary}$ is the restriction of $\mathsf{2N}$ to families of *unary* languages, then

$$\mathsf{2D} \supseteq \mathsf{2N/unary} \impliedby \mathsf{L} \supseteq \mathsf{NL}, \tag{7}$$

where the inclusion on the left is *independent of the lengths of the unary inputs*. Our article has been largely motivated by this recent theorem—and our title reflects this.

2 Preparation

For $n \geq 1$, we let $[n] := \{0, \ldots, n-1\}$. If S is a set, then $\overline{S}$ and $|S|$ are its complement and size. By '$\log(\cdot)$' we always mean '$\log_2(\cdot)$', whereas $\lg(\cdot) := \max(1, \lceil \log_2(\cdot) \rceil)$.

If Σ is an alphabet, then Σ^* is the set of all strings over Σ and ε is the empty string. If x is a string, then $|x|$ is its length, x_i is its ith symbol (for $1 \leq i \leq |x|$), x^i is the concatenation of i copies of x (for $i \geq 0$), and $\langle x \rangle$ is the natural *binary encoding* of x into $|x|$ blocks of $\lg|\Sigma|$ bits each (under a fixed ordering of Σ).

A (*promise*) *problem* over Σ is a pair $\mathfrak{L} = (L, \tilde{L})$ of disjoint subsets of Σ^*. An *instance* of $\mathfrak{L}$ is any $x \in L \cup \tilde{L}$, and is either *positive*, if $x \in L$, or *negative*, if $x \in \tilde{L}$. A machine *solves* $\mathfrak{L}$ if it accepts every positive instance but no negative one. If $\tilde{L} = \overline{L}$ then every $x \in \Sigma^*$ is an instance, and $\mathfrak{L}$ is a *language*.

For $\mathscr{L} = (\mathfrak{L}_h)_{h \geq 1}$ a family of problems and $\mathscr{G}$ a class of functions on the natural numbers, we say that the problems of $\mathscr{L}$ are $\mathscr{G}$*-long* if there exists $g \in \mathscr{G}$ such that $|x| \leq g(h)$, for all $x \in L_h \cup \tilde{L}_h$ and all h. In particular, if $\mathscr{G}$ is the class $\mathrm{poly}(h)$ of all polynomial functions or the class $2^{\mathrm{poly}(h)}$ of all exponential functions, then the problems of $\mathscr{L}$ are respectively *polynomially long* or *exponentially long*. Note that, since every language has arbitrarily long instances, no member $\mathfrak{L}_h$ of a $\mathscr{G}$-long $\mathscr{L}$ can be a language (irrespective of what $\mathscr{G}$ really is).

2.1 Two-Way Finite Automata

A *two-way finite automaton* (2FA, 2NFA, 2DFA) consists of a finite control and an end-marked, read-only input tape, accessed via a two-way head (Fig. 1a). More carefully, a (s, σ)-2NFA is a tuple $A = (S, \Sigma, \delta, q_0, F)$ of a set of states S with $|S| = s$, an input alphabet Σ with $|\Sigma| = \sigma$, a designated start state $q_0 \in S$, a set of designated final states $F \subseteq S$, and a set of transitions

$$\delta \subseteq S \times \big(\Sigma \cup \{\vdash, \dashv\}\big) \times S \times \{\mathrm{L}, \mathrm{R}\},$$

where $\vdash, \dashv \notin \Sigma$ are the two end-markers of the input tape, and L and R are the left and the right directions. An input $x \in \Sigma^*$ is presented on the input tape surrounded by the end-markers, as $\vdash x \dashv$, and is considered *accepted* if δ allows a computation

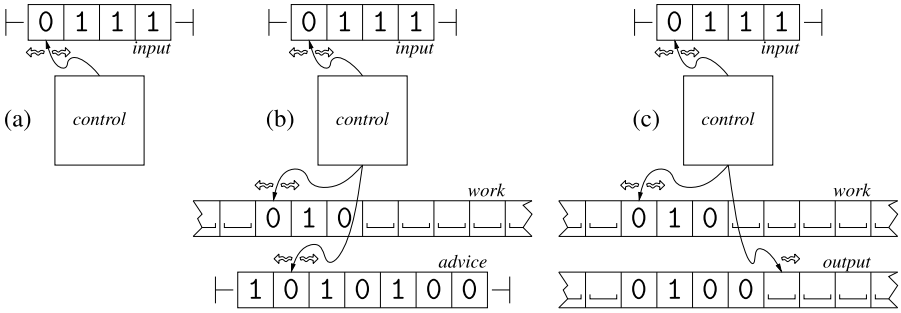

Fig. 1 Types of machines in this article: (**a**) two-way finite automaton, (**b**) Turing machine and (**c**) transducer

that starts at q_0 on $\vdash$ and eventually falls off $\dashv$ into a state $q \in F$.[2] If δ allows at most one computation per input, then A is a 2DFA.

The *behavior* of A on a string x over $\Sigma \cup \{\vdash, \dashv\}$ is the set of tuples (p, d, p', d') of states p, p' and sides $d, d' \in \{\mathrm{L}, \mathrm{R}\}$ such that A can exhibit a computation that starts at p on the d-most symbol of x and eventually falls off the d'-most symbol of x into p'. Easily, the total number of all possible behaviors of A on all possible strings is at most $2^{(2s)^2}$. If A is a 2DFA, then this number is at most $(2s+1)^{2s} = 2^{2s \log(2s+1)}$.

The *language* of A is the set $L(A) := \{x \in \Sigma^* \mid A \text{ accepts } x\}$.

The *binary encoding* of A is the binary string $\langle A \rangle := \mathtt{0}^s\mathtt{1}0^\sigma\mathtt{1}uvw$ where u encodes δ with $2s^2(\sigma+2)$ bits (in the natural way, with 1 bit per possible transition, under some fixed ordering of the set of all possible transitions), whereas v encodes q_0 with $\lg s$ bits and w encodes F with s bits (again in natural ways, under some fixed ordering of S). Easily, $|\langle A \rangle| = O(s^2\sigma)$.

For $\mathscr{A} = (A_h)_{h \geq 1}$ a family of 2NFAs and $\mathscr{F}$ a class of functions on the natural numbers, we say that the automata of $\mathscr{A}$ are *$\mathscr{F}$-large* if there exists $f \in \mathscr{F}$ such that every A_h has $\leq f(h)$ states. In particular, if $\mathscr{F}$ is the class poly(h) of all polynomial functions, then the automata of $\mathscr{A}$ are *small*. We say that $\mathscr{A}$ *solves* a family of problems $\mathscr{L} = (\mathfrak{L}_h)_{h \geq 1}$ if every A_h solves the corresponding $\mathfrak{L}_h$. The class of families of problems that admit $\mathscr{F}$-large 2NFAs is denoted by

$$\mathsf{2NSIZE}(\mathscr{F}) := \{\mathscr{L} \mid \mathscr{L} \text{ is a family of problems that can be solved by a family of } \mathscr{F}\text{-large 2NFAs}\}.$$

Its subclass consisting only of families of $\mathscr{G}$-long problems, for $\mathscr{G}$ a class of functions, is denoted by $\mathsf{2NSIZE}(\mathscr{F})/\mathscr{G}$. (Note that no family in this subclass is a family of languages.) When we omit '$\mathsf{SIZE}(\mathscr{F})$', we mean $\mathscr{F} = \mathrm{poly}(h)$. In particular,

$$\mathsf{2N} := \{\mathscr{L} \mid \mathscr{L} \text{ is a family of problems solvable by a family of small 2NFAs}\}.$$

[2] Note that a set of designated final states F is not really necessary in this definition. Since the 2FA accepts by falling off $\dashv$, designating *all* states as accepting or just *one* state as accepting (e.g., q_0) would have had the same effect. With F, our definition stays comparable to other standard finite automata definitions.

We specifically let $\mathsf{2N/poly} := \mathsf{2N/poly}(h)$ and $\mathsf{2N/exp} := \mathsf{2N}/2^{\mathrm{poly}(h)}$. We also let $\mathsf{2DSIZE}(\mathscr{F})$, $\mathsf{2DSIZE}(\mathscr{F})/\mathscr{G}$, 2D, and $\mathsf{2D}/\mathscr{G}$ be the respective classes for 2DFAs.

As explained in Sect. 1.1, we are interested in the question whether $\mathsf{2D} \supseteq \mathsf{2N}$ and in its bounded variants for exponential or polynomial input lengths, namely the questions whether $\mathsf{2D} \supseteq \mathsf{2N/exp}$ or $\mathsf{2D} \supseteq \mathsf{2N/poly}$. The next lemma explains why we are ignoring super-exponential bounds for the input lengths. The doubly-exponential function used in its statement is just a representative example; it can be replaced by any function that grows faster than every function in $2^{\mathrm{poly}(h)}$.

Lemma 2.1 $\mathsf{2D} \supseteq \mathsf{2N} \iff \mathsf{2D} \supseteq \mathsf{2N}/\{2^{2^h}\}$.

Proof For the interesting direction, suppose $\mathsf{2D} \supseteq \mathsf{2N}/\{2^{2^h}\}$. Pick an arbitrary family of promise problems $\mathscr{L} = (\mathfrak{L}_h)_{h\geq 1} \in \mathsf{2N}$. We will prove that $\mathscr{L} \in \mathsf{2D}$, as well.

Since $\mathscr{L} \in \mathsf{2N}$, we know some family of small 2NFAs $\mathscr{A} = (A_h)_{h\geq 1}$ solves $\mathscr{L}$. For each h, consider the promise problem of checking that a string of at most doubly-exponential length is accepted by A_h:

$$\mathfrak{L}'_h := \left(\left\{x \mid |x| \leq 2^{2^h} \ \&\ x \in L(A_h)\right\},\ \left\{x \mid |x| \leq 2^{2^h} \ \&\ x \in \overline{L(A_h)}\right\}\right).$$

Then the family $\mathscr{L}' := (\mathfrak{L}'_h)_{h\geq 1}$ is in $\mathsf{2N}/\{2^{2^h}\}$, because its problems are doubly-exponentially long and are (obviously) solved by the small 2NFAs of $\mathscr{A}$. Therefore, $\mathscr{L}' \in \mathsf{2D}$ (by our starting assumption). So, some family of small 2DFAs $\mathscr{B} = (B_h)_{h\geq 1}$ solves $\mathscr{L}'$. We will prove that (a modification of) this $\mathscr{B}$ solves $\mathscr{L}$, as well.

Intuitively, the 2DFAs of $\mathscr{B}$ correctly simulate the 2NFAs of $\mathscr{A}$ on all inputs of at most doubly-exponential length. We will argue that, because the 2FAs of both families are small, it is impossible for this simulation to stay correct on inputs of such great length without actually being correct on all possible inputs. So, (a modification of) $\mathscr{B}$ cannot but simulate $\mathscr{A}$ correctly on all inputs, and thus solve $\mathscr{L}$, causing $\mathscr{L} \in \mathsf{2D}$.

For the argument, let us call an index h 'failing' if B_h does not solve $\mathfrak{L}_h$.

Claim *The number of failing h is finite.*

Proof Call an index h 'conflicting' if $L(A_h) \neq L(B_h)$, namely some inputs are accepted by exactly one of A_h and B_h. Then *every failing h is conflicting*. Indeed, if h is failing, then B_h does not solve $\mathfrak{L}_h$; so B_h errs on some instances x, namely

$$x \in L_h \ \&\ B_h \text{ does not accept } x \quad \vee \quad x \in \tilde{L}_h \ \&\ B_h \text{ accepts } x;$$

since A_h solves $\mathfrak{L}_h$, every such instance also satisfies

$$A_h \text{ accepts } x, \text{ but } B_h \text{ does not} \quad \vee \quad A_h \text{ does not accept } x, \text{ but } B_h \text{ does},$$

and is thus in the symmetric difference of $L(A_h)$ and $L(B_h)$, making h conflicting.

So, it suffices to prove finite the number of conflicting h. For this, let a and b be the degrees of the polynomials that bound the sizes of the automata in $\mathscr{A}$ and $\mathscr{B}$, so that every A_h has $O(h^a)$ states and every B_h has $O(h^b)$ states. Then, every A_h and B_h

exhibit respectively $2^{O(h^{2a})}$ and $2^{O(h^b \log h)}$ behaviors. So, the total number l_h of *pairs* of behaviors of A_h and B_h is $2^{O(h^{2a}+h^b \log h)} = 2^{O(h^c)}$, where $c = \max(2a, b+1)$.

Now pick any conflicting h, and let x_h be a *shortest* string in the symmetric difference of $L(A_h)$ and $L(B_h)$. Since B_h solves $\mathfrak{L}'_h$, it is clear that the length of x_h must exceed 2^{2^h}. Less obvious, but also true, is that it cannot exceed l_h. Indeed, if $|x_h| > l_h$, then x_h contains two prefixes on which A_h behaves identically and B_h behaves identically, too. Removing the infix of x_h between the right boundaries of these two prefixes, we get a strictly shorter input y_h that neither A_h nor B_h can distinguish from x_h. Thus, A_h and B_h disagree on y_h iff they disagree on x_h. Given that they disagree on x_h, they must also disagree on y_h, contradicting the selection of x_h as shortest.

Hence, for every conflicting h, our shortest witness of the disagreement satisfies

$$2^{2^h} < |x_h| \le l_h = 2^{O(h^c)}.$$

Since $2^{O(h^c)} = o(2^{2^h})$, only a finite number of h can satisfy these inequalities. □

Now consider the family $\mathscr{B}' = (B'_h)_{h\ge 1}$ that differs from $\mathscr{B}$ only at the failing h, where B'_h is any 2DFA that solves $\mathfrak{L}_h$ (e.g., the standard determinization of A_h [16]). Obviously, $\mathscr{B}'$ solves $\mathscr{L}$. Moreover, if p is a polynomial that bounds the size of the automata of $\mathscr{B}$ and if S is the maximum number of states of a B'_h that corresponds to a failing h, then every B'_h has $\le p(h)$ states, if it is identical to B_h, or $\le S$ states, if h is failing. Hence, the polynomial $p'(h) = p(h) + S$ bounds the size of the automata in $\mathscr{B}'$. Overall, $\mathscr{B}'$ is a family of small 2DFAs solving $\mathscr{L}$. Therefore $\mathscr{L} \in \mathsf{2D}$. □

2.2 Turing Machines

A *Turing machine* (TM, NTM, DTM) is a 2FA with two extra tapes: a bi-infinite, read-write work tape and an end-marked, read-only advice tape, each of which is accessed via a dedicated two-way head (Fig. 1b). More carefully, a (s, σ, γ)-NTM is a tuple $M = (S, \Sigma, \Gamma, \Delta, \delta, q_0, q_f)$ of a set of states S with $|S| = s$, an input alphabet Σ with $|\Sigma| = \sigma$, a work alphabet Γ with $|\Gamma| = \gamma$, an advice alphabet Δ, two designated start and final states $q_0, q_f \in S$, and a set of transitions

$$\delta \subseteq S \times \big(\Sigma \cup \{\vdash, \dashv\}\big) \times \big(\Gamma \cup \{\sqcup\}\big) \times \big(\Delta \cup \{\vdash, \dashv\}\big) \times S \times \Gamma \times \{\mathrm{L},\mathrm{R}\}^3,$$

where $\vdash, \dashv \notin \Sigma \cup \Delta$ are the two end-markers, $\sqcup \notin \Gamma$ is the blank symbol, and L,R are the two directions. An input $x \in \Sigma^*$ and an advice string $y \in \Delta^*$ are presented on the input and advice tapes surrounded by end-markers, as $\vdash x \dashv$ and $\vdash y \dashv$, and are considered accepted if δ allows a computation which starts at q_0, with blank work tape and with the input and advice heads on $\vdash$, and eventually falls off $\dashv$ on the input tape into q_f. If δ allows at most one computation per input, then M is a DTM.

The *internal configuration* of M at a point in the course of its computation is the tuple (q, w, j, i) of its current state q, current non-blank content $w \in \Gamma^*$ on the work tape, current head position j on $\sqcup w \sqcup$, and current head position i on the advice tape.

The *language* of M under a family $\mathscr{y} = (y_m)_{m\geq 0}$ of advice strings over Δ is the set

$$L(M, \mathscr{y}) := \{x \in \Sigma^* \mid M \text{ accepts } x \text{ and } y_{|x|}\}.$$

If $\mathscr{y}$ is the *empty advice* $(\varepsilon)_{m\geq 0}$, then we just write $L(M)$. The *length* of $\mathscr{y}$ is the function $m \mapsto |y_m|$. We say that $\mathscr{y}$ is *strong advice* for M if every y_m is valid not only for all inputs of length m but also for all shorter inputs:

$$\text{for all } x \text{ with } |x| \leq m\text{:} \quad M \text{ accepts } x \text{ and } y_m \iff M \text{ accepts } x \text{ and } y_{|x|}.$$

This deviation from standard definitions of advice is unimportant if the advice length is at least polynomial in m (because then we can always replace y_m with the concatenation of $y_0, y_1, \ldots, y_m$). For shorter advice, though, it is not clear whether there is a difference. Our theorems need the strong variant.

A function f is a *strong space bound* for M under $\mathscr{y}$ if, for all m and all x with $|x| \leq m$, every computation on x and y_m visits at most $f(m)$ work tape cells. If this is guaranteed just for the x and y_m that are accepted and then just for at least one accepting computation, then f is only a *weak space bound* for M under $\mathscr{y}$.

For $\mathscr{F}$ and $\mathscr{G}$ two classes of functions, the class of languages that are solvable by NTMs under strong $\mathscr{G}$-long advice in strongly $\mathscr{F}$-bounded space is denoted by

$$\mathsf{NSPACE}(\mathscr{F})/\mathscr{G} := \{L(M, \mathscr{y}) \mid \mathscr{y} \text{ is strong advice for } M \text{ of length } g \in \mathscr{G} \ \&\ \text{some } f \in \mathscr{F} \text{ is a strong space bound for } M \text{ under } \mathscr{y}\}.$$

When we just write $\mathsf{NSPACE}(\mathscr{F})$, we mean $\mathscr{G} = \{0\}$, so that the only possible advice is the empty one. In particular, we let

$$\mathsf{NL} := \mathsf{NSPACE}(\{\log n\}) \quad \text{and} \quad \mathsf{NLL} := \mathsf{NSPACE}(\{\log\log n\}).$$

The respective classes for DTMs, denoted by $\mathsf{DSPACE}(\mathscr{F})/\mathscr{G}$, $\mathsf{DSPACE}(\mathscr{F})$, L, and LL, are defined analogously. We specifically let $\mathsf{L/poly} := \mathsf{DSPACE}(\{\log n\})/\mathrm{poly}(n)$ and $\mathsf{LL/polylog} := \mathsf{DSPACE}(\{\log\log n\})/\mathrm{poly}(\log n)$.

The next lemma lists the full details of the standard fact that every (advised) TM that computes on inputs of bounded length can be simulated by a 2FA. In the statement, a function f is *fully space constructible* if there exists a DTM which, on any input x and any advice, halts after visiting exactly $f(|x|)$ work tape cells. A *transducer* is a DTM without advice tape but with a write-only output tape, which is accessed via a one-way head (Fig. 1c).

Lemma 2.2 *Consider a* NTM M *under strong advice* $\mathscr{y}$ *of length* g, *obeying a weak space bound* f. *For every length bound* m, *there exists a* 2NFA A_m *with* $2^{O(f(m)+\log g(m))}$ *states which agrees with* M *under* $\mathscr{y}$ *on every input of length at most* m.

Moreover: (a) *If* M *is deterministic*, *then so is* A_m. (b) *If* $\mathscr{y}$ *is empty*, *then* A_m *also agrees with* M *under* $\mathscr{y}$ *on every non-accepted input* (*of any length*). (c) *If* f *is fully space constructible*, *then there exists a transducer* T_M *which*, *given* m (*in unary*) *and the corresponding* y_m, *computes* A_m (*in binary*) *in space* $O(f(m) + \log g(m))$.

Proof Let M be a (s,σ,γ)-NTM and pick $\mathscr{Y} = (y_m)_{m\geq 0}$ and g and f as in the statement. Fix m and consider M working on any input (of any length) and on y_m. Let S_m be the set of all internal configurations of M with $\leq f(m)$ work tape cells when y_m is on the advice tape. The number of such configurations is easily seen to satisfy:

$$\begin{aligned}|S_m| &\leq s\cdot\left(1+\sum_{t=1}^{f(m)}\gamma^t(t+2)\right)\cdot\big(|y_m|+2\big)\\ &\leq s\cdot\big(1+\gamma^{f(m)}\big(f(m)+2\big)f(m)\big)\cdot\big(g(m)+2\big)\\ &= O\big(\gamma^{f(m)}f(m)^2\cdot g(m)\big)=2^{O(f(m)+\log g(m))}.\end{aligned}$$

We let $A_m := (S_m, \Sigma, \delta_m, q_0, F_m)$ where Σ is M's input alphabet, q_0 and F_m are the initial and the accepting internal configurations in S_m, and δ_m contains a transition (c,a,c',d) iff M can change its internal configuration from c to c' and move its input head in direction d when its current input cell contains a and its advice tape contains y_m. (Note that y_m is 'embedded' in δ_m.) This concludes the definition of A_m.

Now fix any input x (of any length). Let τ_m and τ'_m be respectively the computation tree of M on x and y_m and the computation tree of A_m on x. It should be clear that every branch β in τ_m that uses $\leq f(m)$ work tape cells is fully simulated in τ'_m by an equally long branch β', which accepts iff β does. In contrast, every branch β in τ_m that uses $> f(m)$ work tape cells is only partially simulated in τ'_m, by a shorter branch β' which hangs (at a rejecting state). Moreover, every branch of τ'_m is the β' of a branch β of τ_m under the correspondence established by the above two cases.

Now suppose $|x| = n \leq m$. We must prove that $x \in L(M,\mathscr{Y})$ iff A_m accepts x. For this, let τ_n be the computation tree of M on x and y_n, and note that τ_n accepts iff τ_m does (because $\mathscr{Y}$ is strong). We distinguish the natural two cases.

- If $x \in L(M,\mathscr{Y})$, then τ_n accepts. Hence, τ_m accepts as well. Therefore, some accepting branch β in τ_m uses $\leq f(m)$ work tape cells (because f is a weak space bound). So, the corresponding branch β' in τ'_m completes the simulation and accepts, too. Hence, A_m accepts x.
- If $x \notin L(M,\mathscr{Y})$, then τ_n does not accept. Hence, τ_m does not accept either. Therefore, every branch in τ'_m is non-accepting, either because it fully simulates a non-accepting branch of τ_m or because it hangs after partially simulating one. Hence, A_m does not accept x.

Overall, A_m agrees with M under $\mathscr{Y}$ on every x of length $n \leq m$, as required. We continue with the remaining, special considerations of the lemma.

(a) *If M is deterministic*, then A_m is clearly also deterministic.
(b) *If $\mathscr{Y}$ is empty*, then the agreement between A_m and M under $\mathscr{Y}$ extends also to every $x \notin L(M,\mathscr{Y})$ of length $n > m$. Indeed: Since $x \notin L(M,\mathscr{Y})$, we know τ_n does not accept, hence τ_m does not accept (because, since $\mathscr{Y}$ is empty, we have $y_n = y_m$ and thus $\tau_n = \tau_m$), and thus τ'_m does not accept (as in the second case above).[3]

[3]Note that, in contrast, for an $x \in L(M,\mathscr{Y})$ with $n > m$ we have no guarantee that A_m agrees with M under $\mathscr{Y}$. It may be that the computation tree τ_n of M on input x and advice $y_n = \varepsilon$ uses $f(n)$ work

(c) *If f is fully space constructible*, then the binary encoding $\langle A_m \rangle$ can be computed from 0^m and y_m, as follows. We first mark exactly $f(m)$ work tape cells, by running on 0^m the DTM that fully constructs f. Then, using these cells and y_m, we mark another $\lg s + \lg(f(m)+2) + \lg(|y_m|+2)$ cells. Now the marked region is as long as the longest description of an internal configuration $c \in S_m$. This makes it possible to iterate over all $c \in S_m$ or pairs thereof (by lexicographically iterating over all possible strings and discarding those that are non-descriptions). From this point on, computing the bits of $\langle A_m \rangle$ is fairly straightforward. Overall, we will need no more than $O(f(m) + \log g(m))$ work tape cells. □

Corollary 2.1

(a) *For every* NTM M *(under empty advice) of weak space* $O(\log n)$ *and for every length bound* m*, there exists a* $\mathrm{poly}(m)$*-state* 2NFA A_m *that may disagree with* M *only on accepted inputs longer than* m*—and is deterministic, if* M *is. Moreover, there exists a logarithmic-space transducer* T_M *which, given* m *(in unary), computes* A_m *(in binary).*

(b) *For every* NTM M *(under empty advice) of weak space* $O(\log \log n)$ *and for every length bound* m*, there exists a* $\mathrm{poly}(\log m)$*-state* 2NFA A_m *that may disagree with* M *only on accepted inputs longer than* m*—and is deterministic, if* M *is.*

Proof

(a) Let T_M be the transducer guaranteed by Lemma 2.2 for M under the empty advice and the fully space constructible space bound $f(m) = \log m$. Since the advice is empty, we know that the second input to T_M is always ε, that the advice length is $g(m) = 0$, and that A_m also agrees with M on every non-accepted input. Therefore, T_M works on the single input 0^m, uses space $O(f(m) + \log g(m)) = O(\log m)$, and outputs a 2NFA that has $2^{O(f(m)+\log g(m))} = \mathrm{poly}(m)$ states and may disagree with M only on accepted inputs longer than m.

(b) Let A_m be the 2NFA guaranteed by Lemma 2.2 for m and for M under the empty advice and the weak space bound $f(m) = O(\log \log m)$. Since the advice is empty, we know that its length is $g(m) = 0$ and that A_m also agrees with M on all non-accepted inputs. Therefore, A_m has $2^{O(f(m)+\log g(m))} = 2^{O(\log \log m)} = \mathrm{poly}(\log m)$ states and may disagree with M only on accepted inputs longer than m. □

2.3 Reductions

Let Σ_1 and Σ_2 be two alphabets, neither of which contains the end-markers $\vdash$ and $\dashv$. Every function $r : \Sigma_1 \cup \{\vdash, \dashv\} \to \Sigma_2^*$ extends naturally to the entire $\vdash \Sigma_1^* \dashv$ by the rule

$$r(\vdash x \dashv) := r(\vdash) r(x_1) \cdots r(x_{|x|}) r(\dashv)$$

tape cells on every accepting branch. Hence, if $f(n) > f(m)$, our A_m will be missing some of the states necessary to simulate these accepting branches in its computation tree τ'_m on x. Therefore, τ_n will be accepting but τ'_m will be rejecting—despite the fact that $y_n = y_m = \varepsilon$.

for every $x \in \Sigma_1^*$. Such a function is called a *homomorphic reduction* of a problem $\mathfrak{L}_1 = (L_1, \tilde{L}_1)$ over Σ_1 to a problem $\mathfrak{L}_2 = (L_2, \tilde{L}_2)$ over Σ_2 if it satisfies

$$x \in L_1 \implies r(\vdash x \dashv) \in L_2 \quad \text{and} \quad x \in \tilde{L}_1 \implies r(\vdash x \dashv) \in \tilde{L}_2,$$

for all x [13]. When such a reduction exists, we say that $\mathfrak{L}_1$ *homomorphically reduces to* $\mathfrak{L}_2$, and write $\mathfrak{L}_1 \leq_{\mathrm{h}} \mathfrak{L}_2$. The *expansion* of r is the function which maps every length bound m to the maximum length of the r-image of (the end-marked extension of) a string of length $\leq m$:

$$\mu_r(m) := \max\left\{ \left| r(\vdash x \dashv) \right| \mid x \in \Sigma_1^* \text{ and } |x| \leq m \right\}.$$

The *ternary encoding* of r, assuming a fixed ordering $a_1, \ldots, a_{\sigma_1}$ of Σ_1, is

$$\langle r \rangle := \left\langle r(\vdash) \right\rangle \# \left\langle r(a_1) \right\rangle \# \cdots \# \left\langle r(a_{\sigma_1}) \right\rangle \# \left\langle r(\dashv) \right\rangle.$$

(On the right-hand side of this definition, $\langle \cdot \rangle$ stands for the standard binary encoding of strings over Σ_2, as explained in the beginning of Sect. 2; so $\langle r \rangle \in \{0, 1, \#\}^*$.)

The next lemma lists the full details of the old fact that the class of problems that can be solved by small 2FAs is closed under homomorphic reductions [13].

Lemma 2.3 *If* $\mathfrak{L}_1 \leq_{\mathrm{h}} \mathfrak{L}_2$ *and some* s*-state* 2NFA A_2 *solves* $\mathfrak{L}_2$, *then some* $2s$*-state* 2NFA A_1 *solves* $\mathfrak{L}_1$.

If A_2 *is deterministic, then so is* A_1. *Moreover, there exists a logarithmic-space transducer* T_{h} *which, given a homomorphic reduction of* $\mathfrak{L}_1$ *to* $\mathfrak{L}_2$ (*in ternary*) *and a deterministic* A_2 (*in binary*), *computes* A_1 (*in binary*).

Proof Suppose $r : \Sigma_1 \cup \{\vdash, \dashv\} \to \Sigma_2^*$ is a homomorphic reduction of $\mathfrak{L}_1 = (L_1, \tilde{L}_1)$ to $\mathfrak{L}_2 = (L_2, \tilde{L}_2)$, and the 2NFA $A_2 = (S_2, \Sigma_2, \delta_2, q_2, F_2)$ solves $\mathfrak{L}_2$ with $|S_2| = s$.

On input $x \in \Sigma_1^*$, the new automaton A_1 simulates A_2 on $r(\vdash x \dashv)$ piece-by-piece: on each symbol a on its tape, A_1 does exactly what A_2 would eventually do on the corresponding infix $r(a)$ on its own tape—except if a is $\vdash$ or $\dashv$, in which case the corresponding infix is respectively the prefix $\vdash r(\vdash)$ or the suffix $r(\dashv)\dashv$. This way, every branch β in the computation tree τ_2 of A_2 on $r(\vdash x \dashv)$ is simulated by a branch in the computation tree τ_1 of A_1 on x, which is accepting iff β is; in addition, these simulating branches are the only branches in τ_1. Thus, τ_1 contains accepting branches iff τ_2 does, namely A_1 accepts x iff A_2 accepts $r(\vdash x \dashv)$. Consequently, if $x \in L_1$, then $r(\vdash x \dashv) \in L_2$ (by the selection of r), hence A_2 accepts, and thus A_1 accepts as well. In contrast, if $x \in \tilde{L}_1$, then $r(\vdash x \dashv) \in \tilde{L}_2$ (again by the selection of r), hence A_2 does not accept, and thus A_1 does not accept either. Overall, A_1 solves $\mathfrak{L}_1$.

To perform this simulation, A_1 keeps track of the current state of A_2 and the side (L or R) from which A_2 enters the current corresponding infix. Formally, $A_1 := (S_2 \times \{\mathrm{L}, \mathrm{R}\}, \Sigma_1, \delta_1, (q_2, \mathrm{L}), F_2 \times \{\mathrm{L}\})$, where δ_1 is fairly straightforward to derive from the above informal description. For example, $((p, \mathrm{L}), a, (q, \mathrm{L}), \mathrm{R}) \in \delta_1$ iff δ_2 allows on $r(a)$ a computation that starts at p on the leftmost symbol and eventually falls off the rightmost boundary into q (thus entering the subsequent infix from its left side). For another example, $((p, \mathrm{L}), \dashv, (q, \mathrm{R}), \mathrm{L}) \in \delta_1$ iff δ_2 allows on $r(\dashv)\dashv$ a computation

that starts at p on the leftmost symbol and eventually falls off the leftmost boundary into q (thus entering the preceding infix from its right side).

If A_2 is deterministic, then clearly A_1 is also deterministic. Moreover, its encoding $\langle A_1 \rangle$ can then be computed from the two encodings $\langle r \rangle$ and $\langle A_2 \rangle$ in deterministic logarithmic space. To see how, let $\sigma_1 := |\Sigma_1|$ and $\sigma_2 := |\Sigma_2|$, and recall that

$$\langle A_1 \rangle = \mathtt{0}^{2s}\mathtt{1}\mathtt{0}^{\sigma_1}\mathtt{1}u_1v_1w_1 \quad \text{and} \quad \langle A_2 \rangle = \mathtt{0}^{s}\mathtt{1}\mathtt{0}^{\sigma_2}\mathtt{1}u_2v_2w_2,$$

where the u_1, v_1, w_1 and u_2, v_2, w_2 encode respectively the transition functions, the start states, and the sets of final states. Each part of $\langle A_1 \rangle$ can be computed in a straightforward manner from the corresponding part of $\langle A_2 \rangle$ (using no work tape at all), except for $\mathtt{0}^{\sigma_1}$ and u_1. For the former, we scan $\langle r \rangle$ and output a $\mathtt{0}$ for every occurrence of # after the first one. For the latter, we mark $2\lg s + \lg(\sigma_1+2) + 3 = O(\log(\sigma_1 s))$ work tape cells and use them to iterate over all encodings of tuples of the form

$$\big((p,d),a,\big(p',d'\big),d''\big) \quad \text{for } p, p' \in [s] \text{ and } a \in \Sigma_1 \cup \{\vdash,\dashv\} \text{ and } d, d', d'' \in \{\mathrm{L},\mathrm{R}\},$$

outputting 1 bit per tuple. To decide each bit, we locate $\langle r(a) \rangle$ in $\langle r \rangle$ and simulate A_2 on $r(a)$ (or on $\vdash r(a)$ if $a = \vdash$, or on $r(a)\dashv$ if $a = \dashv$), starting at p on the d-most symbol and counting the simulated steps. We continue until either we fall off the string or the number of simulated steps exceeds $s \cdot |r(a)| = s \cdot |\langle r(a) \rangle| / \lg \sigma_2$ by 1 (or by $s+1$, if $a = \vdash$ or $\dashv$). If we fall off the d''-most boundary into p' and $d' \neq d''$ then we output $\mathtt{1}$; in all other cases, we output $\mathtt{0}$. This simulation needs only a finite number of pointers into $\langle r \rangle$ and u_2, plus the aforementioned bounded counter. Overall, the space used is logarithmic in $|\langle r \rangle + \langle A_2 \rangle|$. □

2.4 Two-Way Liveness

Central in our arguments in Sect. 3 are several variants of a computational problem that we call *two-way liveness* [13]. In this section we introduce these variants and study their properties.

Fix any $h \geq 1$. Let Γ_h be the alphabet of all possible directed graphs consisting of two columns of h nodes each (Fig. 2a). Easily, $|\Gamma_h| = 2^{(2h)^2}$. Every string $x \in \Gamma_h^*$ of such graphs induces the multicolumn graph that we get when we identify adjacent columns (Fig. 2b). If this h-tall $(|x|+1)$-column graph contains a path from its leftmost to its rightmost column, we say that x is *live*; otherwise, no such path exists and we say that x is *dead*. The language

$$\textsc{two-way liveness}_h = \textsc{twl}_h := \big\{x \in \Gamma_h^* \mid x \text{ is live}\big\}$$

represents the computational problem of checking that a given string of h-tall two-column graphs is live. The family of languages

$$\textsc{two-way liveness} = \textsc{twl} := (\textsc{twl}_h)_{h \geq 1}$$

represents this computational task for all possible heights h.

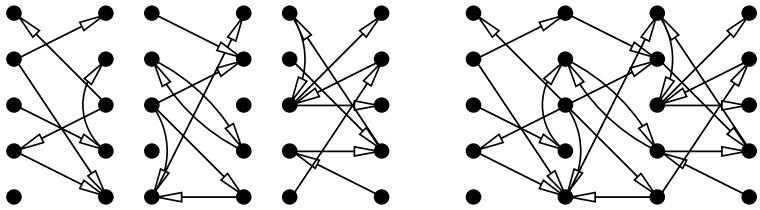

Fig. 2 (**a**) Three symbols of Γ_5, and (**b**) the multicolumn graph defined by them

The next lemma lists the full details of the old fact that every problem that can be solved by a 2NFA reduces homomorphically to the problem of checking two-way liveness on graphs of height twice the number of states in the automaton [13].

Lemma 2.4 *If $\mathfrak{L}$ can be solved by an s-state* 2NFA *A, then $\mathfrak{L} \leq_{\mathrm{h}} \mathrm{TWL}_{2s}$ via a reduction which has expansion $m+2$ and is constructible from $\langle A \rangle$ in logarithmic space.*

Proof Let $A = (S, \Sigma, \delta, q_0, F)$ with $|S| = s$. We consider a reduction r which maps every $a \in \Sigma \cup \{\vdash, \dashv\}$ to a single symbol $r(a) \in \Gamma_{2s}$. This is the unique symbol that fully encodes the behavior of A on a, under the following scheme.

First, in each of the two columns of $r(a)$ we distinguish the upper and the lower half. We thus get four half-columns (upper-left, lower-left, upper-right, and lower-right) of s nodes each, representing four copies of S. For every tuple (p, d, p', d') in the behavior of A on a we add an arrow with origin the p-th node of either the upper-left or the lower-right half-column, depending on whether $d = \mathrm{L}$ or R, and destination the p'-th node of the lower-left or the upper-right half-column, depending on whether $d' = \mathrm{L}$ or R. In the special case when $a = \vdash$, an arrow with origin in the upper-left half-column is added only if $p = q_0$. In the special case when $a = \dashv$, an arrow with destination in the upper-right half-column is added only if $p' \in F$.

With these definitions, it is not hard to see that the computation tree of A on a string $x \in \Sigma^*$ contains an accepting branch iff $r(\vdash x \dashv)$ contains a path from its leftmost to its rightmost column (see [13] for details). Hence, r is a correct reduction of the entire $L(A)$ to TWL_{2s}, and thus also of $\mathfrak{L}$ to TWL_{2s}. Obviously, $|r(\vdash x \dashv)| = |\vdash x \dashv| = |x| + 2$ for every $x \in \Sigma^*$, and thus the expansion is $\mu(m) = m + 2$. Finally, constructing $\langle r \rangle$ out of $\langle A \rangle$ reduces to computing the behavior of A on each $a \in \Sigma \cup \{\vdash, \dashv\}$, which is straightforward to do with a finite number of pointers into $\langle A \rangle$. □

In order to establish Theorem 1.1, we will need a single language that can encode the problem of checking two-way liveness on every possible height. To introduce this language, we start with the *binary encoding* $\langle a \rangle$ of a single graph $a \in \Gamma_h$, which is a string of $(2h)^2$ bits that describes a in the natural way (with 1 bit per possible arrow, under a fixed ordering of the set of all possible arrows). Using this encoding scheme, we can "join" all languages of the family TWL into the one binary language:

$$\mathrm{TWL\ JOIN} := \big\{ 0^h 1 \langle a_1 \rangle \langle a_2 \rangle \cdots \langle a_l \rangle 1 0^t \mid h \geq 1 \ \&\ h \text{ divides } t \ \&\ l \geq 0 \ \&\ \text{each } a_i \in \Gamma_h \ \&\ a_1 a_2 \cdots a_l \text{ is live} \big\}.$$

The interpretation is driven by the h leading 0s: the leftmost 1 and the rightmost 1 must be separated by 0 or more blocks of bits of length $(2h)^2$ each (the "middle bits"); the multicolumn graph encoded by these blocks must contain a path from its leftmost to its rightmost column; and the number t of trailing 0s must be a multiple of h.[4]

In order to establish Theorem 1.3, we will need a variant of TWL JOIN, called TWL LONG-JOIN. This differs only in the specification for the number of trailing 0s, which must now be a multiple of every positive $j \leq h$, as opposed to just h:

$$\text{TWL LONG-JOIN} := \big\{ 0^h 1 \langle a_1 \rangle \langle a_2 \rangle \cdots \langle a_l \rangle 1 0^t \mid h \geq 1 \ \&\ \text{all } j \leq h \text{ divide } t \ \&\ l \geq 0 \ \&\ \text{each } a_i \in \Gamma_h \ \&\ a_1 a_2 \cdots a_l \text{ is live} \big\}.$$

Here, we are copying the padding scheme used by Szepietowski in [18]. In short, this scheme pads a string of length h with a string of 0s whose length is (i) at least exponential in h and (ii) checkable in space logarithmic in h—and thus log-logarithmic in the new total length, hence its usefulness for Theorem 1.3.

Finally, the proof of Theorem 1.2 requires yet more variants of TWL JOIN. For every $k \geq 1$, we let TWL JOIN_k be the variant where the number of trailing 0s must be a multiple of every positive $j \leq \lfloor \log h \rfloor^k$. This leads to a padding string whose length is (i) at least quasi-polynomial in h and (ii) checkable in space poly-logarithmic in h. (If $k = 1$ then the length is at least polynomial and the space logarithmic, causing the properties of TWL JOIN_1 to be similar to those of TWL JOIN.)

The next two lemmas study the properties of these variants of two-way liveness. They also discuss another problem, the *acceptance problem* for 2NFAs, with which it will be useful to compare in Sect. 3. This is the problem of checking whether a given 2NFA accepts a given input, formally the ternary language

$$\text{A}_{\text{2NFA}} := \big\{ \langle A \rangle \# \langle x \rangle \mid A \text{ is a 2NFA and } A \text{ accepts } x \big\}.$$

Note that the alphabet of the given automaton is not fixed: to solve this problem, one must also check that the binary string after # can indeed be interpreted as an input for the 2NFA that is encoded by the binary string before #. Finally, we also let A_{2DFA} be the corresponding problem for 2DFAs.

Lemma 2.5 *If $\mathfrak{L}$ can be solved by a (s, σ)-2NFA A, then $\mathfrak{L}$ homomorphically reduces to each of* A_{2NFA}, TWL JOIN, TWL JOIN_k, *and* TWL LONG-JOIN *via reductions of expansion $O(s^2 \sigma m)$, $O(s^2 m)$, $2^{O(\log^k s)} m$, and $2^{O(s)} m$, respectively. The first two reductions are constructible from $\langle A \rangle$ in logarithmic space.*[5]

[4] These trailing 0s are, in fact, redundant. They are included in the definition just for symmetry with the definitions of the variants in the next two paragraphs.

[5] More tightly, the expansion of the first reduction is $O(s^2 \sigma) + m \lg \sigma$. But the looser $O(s^2 \sigma m)$ is simpler and does not harm our conclusions, since it is later (Theorem 3.0, forward direction) fed to an unspecified polynomial. Similar looseness is adopted elsewhere, too.

Proof Let the alphabet of A be $\Sigma = \{a_1, \ldots, a_\sigma\}$. We construct four homomorphic reductions of $\mathfrak{L}$ to the four languages of the statement, and call them respectively

$$r_0 : \Sigma \cup \{\vdash, \dashv\} \to \{0, 1, \#\}^* \quad \text{and} \quad r_1, r_2, r_3 : \Sigma \cup \{\vdash, \dashv\} \to \{0, 1\}^*.$$

It will be straightforward to verify that each reduction is correct, and that $\langle r_0 \rangle$ and $\langle r_1 \rangle$ can be constructed from $\langle A \rangle$ in logarithmic space, so we will omit these discussions.

Reduction r_0 maps $\vdash$ to $\langle A \rangle \#$, every a_i to the $\lg \sigma$-long binary representation of $i-1$, and $\dashv$ to the empty string. As a result, every string x of length $n \le m$ is transformed into the string $r_0(\vdash x \dashv) = \langle A \rangle \# \langle x \rangle$, of length $(|\langle A \rangle| + 1) + n \lg \sigma + 0 = O(s^2 \sigma) + n \lg \sigma = O(s^2 \sigma m)$.

Reduction r_1 uses the reduction $r : \Sigma \cup \{\vdash, \dashv\} \to \Gamma_{2s}$ given by Lemma 2.4, along with the binary encoding $\langle \cdot \rangle$ of the graphs of Γ_{2s}. It maps $\vdash$ to $0^{2s} 1 \langle r(\vdash) \rangle$, every a_i to $\langle r(a_i) \rangle$, and $\dashv$ to $\langle r(\dashv) \rangle 1 0^{2s}$. As a result, every string x of length $n \le m$ is transformed into the string

$$r_1(\vdash x \dashv) \;=\; 0^{2s} 1 \big\langle r(\vdash) \big\rangle \big\langle r(x_1) \big\rangle \cdots \big\langle r(x_n) \big\rangle \big\langle r(\dashv) \big\rangle 1 0^{2s},$$

which has length $(2s+1) + (n+2)(4s)^2 + (1+2s) = O(s^2 m)$ and is in TWL JOIN iff $r(\vdash) r(x_1) \cdots r(x_n) r(\dashv)$ is live (all other conditions in the definition of TWL JOIN are obviously satisfied), namely iff $r(\vdash x \dashv) \in \mathrm{TWL}_{2s}$.

The last two reductions, r_2 and r_3, use the function λ defined by

$$\text{for every } l \ge 1: \quad \lambda(l) := \mathrm{lcm}\{1, 2, \ldots, l\},$$

which is known to satisfy $\lambda(l) = 2^{\Theta(l)}$ [18, Lemma, part (b)]. The two reductions differ from r_1 only in that they map $\dashv$ respectively to

$$\big\langle r(\dashv) \big\rangle 1 0^{\lambda(\lfloor \log 2s \rfloor^k)} \quad \text{and} \quad \big\langle r(\dashv) \big\rangle 1 0^{\lambda(2s)}.$$

Hence, every string x of length $n \le m$ is now mapped to the string $r_2(\vdash x \dashv)$ of length

$$(2s+1) + (n+2)(4s)^2 + \big(1 + 2^{\Theta(\lfloor \log 2s \rfloor^k)}\big) = 2^{O(\log^k s)} m,$$

and to the string $r_3(\vdash x \dashv)$ of length $2^{O(s)} m$ (by a similar calculation). □

Lemma 2.6 *The problems* $\mathrm{A}_{\mathrm{2NFA}}$, TWL JOIN, $\mathrm{TWL\ JOIN}_k$ *and* TWL LONG-JOIN *are respectively* NL*-complete*, NL*-complete*, *in* $\mathsf{NSPACE}(\sqrt[k]{\log n})$, *and in* NL.

Proof That $\mathrm{A}_{\mathrm{2NFA}}$ is NL-complete is well-known, so we focus on the other claims.

To prove that TWL JOIN is NL-hard, pick any L recognized by a NTM M in logarithmic space. Every instance x of L can be transformed into an instance of TWL JOIN by the following series of transductions, where $m = |x|$:

$$x \longrightarrow x \# 0^m \xrightarrow{T_M} x \# \langle A_m \rangle \longrightarrow x \# \langle r \rangle \longrightarrow r(\vdash x \dashv).$$

First, we append $\#0^m$ by running on x an obvious zero-space transducer. Then, we replace 0^m with the encoding of the 2NFA A_m guaranteed by Corollary 2.1(a) for M

and m, by running on $\mathtt{0}^m$ the logarithmic-space transducer T_M given by that corollary. Next, we replace $\langle A_m \rangle$ with the encoding of the reduction r guaranteed by Lemma 2.5 for $\mathfrak{L} = L(A_m)$ and TWL JOIN, by running on $\langle A_m \rangle$ the corresponding logarithmic-space transducer given by that lemma. Finally, we replace $x\#\langle r \rangle$ with $r(\vdash x \dashv)$ by running on it the obvious logarithmic-space transducer that applies r to every symbol of $\vdash x \dashv$. By the transitivity of logarithmic-space transductions, it follows that the full transformation can also be implemented in logarithmic space. Moreover, by the selection of M, m, A_m, and r, we have

$$x \in L \iff M \text{ accepts } x \iff x \in L(A_m) \iff r(\vdash x \dashv) \in \text{TWL JOIN}.$$

Therefore, L reduces to TWL JOIN in logarithmic space, as required.

To solve TWL JOIN by a NTM in logarithmic space, we work in two stages. First, we deterministically verify the format: we check that there are at least two `1`s, compute the number h of leading `0`s, and check that h divides the number of trailing `0`s (by scanning and counting modulo h) and that $(2h)^2$ divides the number of "middle bits" (similarly). Then, we nondeterministically verify liveness, by simulating on the "middle bits" the $2h$-state 2NFA solving TWL_h [13]. Easily, each stage can be performed in space $O(\log h)$. Since h is obviously smaller than the total input length n, we conclude that the space usage is indeed $O(\log n)$. Thus TWL JOIN $\in$ NL, completing the proof that TWL JOIN is NL-complete.

To solve TWL LONG-JOIN by a NTM in log-logarithmic space, we use the same algorithm as for TWL JOIN, but with a preliminary stage [18]. This starts by checking that there exist at least two `1`s; if not, we reject. Then, we increment a counter from 1 up to the first number, t^*, that does not divide the number t of trailing `0`s (as above, divisibility is always tested by scan-and-count). On reaching t^*, we compare it with the number h of leading `0`s (scan-and-count, again). If $t^* \leq h$, we reject; otherwise we continue to the two main stages (omitting, however, the check that h divides t). We consider the correctness of this algorithm to be clear, and discuss only the space usage. First, in the preliminary stage the cost is clearly $O(\log t^*)$. Then, as above, the two main stages cost $O(\log h)$, which is also $O(\log t^*)$ because we get to them only if $h < t^*$. Since $t^* = O(\log t)$ [18, Lemma, part (d)] and $t \leq n$ (obviously), we conclude that the total space used is $O(\log \log n)$, as required.

To solve TWL JOIN_k by a NTM in space $O(\sqrt[k]{\log n})$, we use the same algorithm as for TWL LONG-JOIN, but with a modification. Instead of testing whether $t^* \leq h$, we test whether $\sqrt[k]{t^*} \leq \lfloor \log h \rfloor$. If so, then $t^* \leq \lfloor \log h \rfloor^k$, hence the padding length is incorrect, and we reject; otherwise, the padding length is correct and we proceed to the two main stages. To test whether $\sqrt[k]{t^*} \leq \lfloor \log h \rfloor$, we test the equivalent condition $\lceil \sqrt[k]{t^*} \rceil \leq \lfloor \log h \rfloor$. For this, we first compute $\tilde{t} := \lceil \sqrt[k]{t^*} \rceil$ from t^* by standard methods in space $O(\log t^*)$, then test $\tilde{t} \leq \lfloor \log h \rfloor$. This last test can be performed using two counters α and β, as follows. After initializing $\alpha := \tilde{t}$ and $\beta := 0$, we start scanning the prefix $\mathtt{0}^h\mathtt{1}$ of the input, incrementing β at every step and decrementing α at every change of the *length* of β, until either α becomes 0 or we reach `1` on the input tape.

- *If the former condition is met first*, then the initial value $\tilde{t}$ of α equals the actual number of times that the length of β increased, and is thus bounded by the maximum number of times $\lfloor \log h \rfloor$ that the length of β can possibly increase (in the

case when the latter condition is met first). That is, $\tilde{t} \leq \lfloor \log h \rfloor$ and we reject. The space usage, for computing $\tilde{t}$, for storing α, and for storing β, is respectively:

$$O\big(\log t^*\big) + O(\log \tilde{t}) + (\tilde{t}+1) = O\big(\sqrt[k]{t^*}\big) = O(\sqrt[k]{\log t}) = O(\sqrt[k]{\log n}).$$

The first equality is clear, then we use that $t^* = O(\log t)$ and $t \leq n$ (as above).

- *If the latter condition is met first*, then $\tilde{t} > \lfloor \log h \rfloor$ and we continue to the two main stages. The space usage is again $O(\log t^*) + O(\log \tilde{t})$ for computing $\tilde{t}$ and storing α, plus $\lfloor \log h \rfloor + 1$ for storing β and $O(\log h)$ for the main stages. Overall:

$$O\big(\log t^*\big) + O(\log \tilde{t}) + \big(\lfloor \log h \rfloor + 1\big) + O(\log h) = O\big(\sqrt[k]{t^*}\big) = O(\sqrt[k]{\log n}).$$

The first equality uses the fact that $\lfloor \log h \rfloor < \tilde{t}$, while the second one uses the fact that $t^* = O(\log t)$ and $t \leq n$ (as above).

Overall, in both cases, the total space usage is $O(\sqrt[k]{\log n})$, as required. □

3 The Berman-Lingas Theorem

The 'Berman-Lingas Theorem' is Theorem 6 of the old technical report [3]. It is usually cited as follows:

> If $\mathsf{L} = \mathsf{NL}$ then for every s-state 2NFA there exists a poly(s)-state 2DFA that agrees with it on all poly(s)-long inputs. (8)

However, the full statement that was actually claimed in that report, via a remark a few pages after the statement of Theorem 6, is much stronger:[6]

> $\mathsf{L} = \mathsf{NL}$ iff for every alphabet Σ there exists a logarithmic-space transducer T_Σ which, on input a 2NFA A over Σ (in binary) and a length bound m (in unary), outputs a 2DFA B (in binary) that has at most poly(sm) states, for s the number of states in A, and may disagree with A only on accepted inputs longer than m. (8^+)

That is, the report also claimed that the promised 2DFA is constructible from the 2NFA and the length bound in logarithmic space, and that then the converse is also true. The usual citation (8) is just a weak corollary of (8^+) for the case when $m = \text{poly}(s)$.

Unfortunately, the proofs of the above statements depend on the fact that $|\Sigma|$ is constant. In cases where the alphabet grows with s (as is often true in the study of 2D versus 2N), the bound that is guaranteed by these proofs for the number of states in B may very well be exponential in s. To highlight this subtle point, we state and prove the theorem under no assumptions for the alphabet size.

[6] The actual statement of [3, Theorem 6] is very close to the usual citation (8); however, it also includes a pointer to a Remark a few pages later (p. 17), which explains that the promised 2DFA can be constructed in logarithmic space and that with this observation the theorem becomes an equivalence. We also note that the actual statement of [3, Theorem 6] uses $s \cdot m$ as the length bound, for s the number of states in the 2NFA, whereas the full statement (8^+) uses just m; the two statements are equivalent, but (8^+) is simpler and facilitates comparison with subsequent statements.

Theorem 3.0 (Berman and Lingas [3]) *The following statements are equivalent*:

(A) $\mathsf{L} \supseteq \mathsf{NL}$.

(B) *There exists a logarithmic-space transducer T which, on input a* 2NFA A *(in binary) and a length bound m (in unary), outputs a* 2DFA B *(in binary) that has at most* $\mathrm{poly}(s\sigma m)$ *states, for s and σ the number of states and symbols in A, and may disagree with A only on accepted inputs longer than m.*

Proof

[(A)⇒(B)] Suppose $\mathsf{L} \supseteq \mathsf{NL}$. Then $\mathrm{A}_{\mathrm{2NFA}} \in \mathsf{L}$. Therefore, some DTM M under empty advice solves $\mathrm{A}_{\mathrm{2NFA}}$ in space strongly bounded by $\log n$.

To see the main idea behind the argument, consider any (s,σ)-2NFA A and any length bound m. Lemma 2.5 says that $L(A) \leq_{\mathrm{h}} \mathrm{A}_{\mathrm{2NFA}}$ via a reduction of expansion μ. Hence, Lemma 2.3 implies that constructing a 2DFA B for $L(A)$ and input lengths $\leq m$ reduces to constructing a 2DFA $\tilde{B}$ for $\mathrm{A}_{\mathrm{2NFA}} = L(M)$ and input lengths $\leq \mu(m)$. This latter construction is indeed possible by Corollary 2.1(a).

So, given A and m, we employ a series of logarithmic-space transductions:

$$\langle A\rangle \# 0^m \longrightarrow \langle A\rangle \# 0^{\mu(m)} \xrightarrow{T_M} \langle A\rangle \# \langle \tilde{B}\rangle \longrightarrow \langle r\rangle \# \langle \tilde{B}\rangle \xrightarrow{T_{\mathrm{h}}} \langle B\rangle.$$

First, we replace 0^m with $0^{\mu(m)}$, by running on $\langle A\rangle \# 0^m$ an obvious logarithmic-space transducer which performs the algebra for $\mu(m)$ on the parameters of A and on m. Then we replace $0^{\mu(m)}$ with the encoding of the 2DFA $\tilde{B}$ guaranteed by Corollary 2.1(a) for M and for $\mu(m)$, by running on $0^{\mu(m)}$ the logarithmic-space transducer T_M given by that corollary. Then we replace $\langle A\rangle$ with the encoding of the homomorphism r guaranteed by Lemma 2.5 for $L(A) \leq_{\mathrm{h}} \mathrm{A}_{\mathrm{2NFA}}$, by running on $\langle A\rangle$ the logarithmic-space transducer given by that lemma. Finally, we convert $\langle r\rangle \# \langle \tilde{B}\rangle$ into the desired 2DFA B, by running the logarithmic-space transducer T_{h} given by Lemma 2.3. By the transitivity of logarithmic-space transductions, we know that the full algorithm can also be implemented in logarithmic space. It remains to verify the properties of B.

If S and $\tilde{S}$ are the sets of states in B and $\tilde{B}$, then $|S| = 2|\tilde{S}|$ (by Lemma 2.3) and $|\tilde{S}| = \mathrm{poly}\big(\mu(m)\big)$ (by Corollary 2.1(a)) and $\mu(m) = O(s^2\sigma m)$ (by Lemma 2.5). So,

$$|S| = 2|\tilde{S}| = 2 \cdot \mathrm{poly}\big(\mu(m)\big) = 2 \cdot \mathrm{poly}\big(O\big(s^2\sigma m\big)\big) = \mathrm{poly}(s\sigma m),$$

as promised. Moreover, consider any input x to A. If $y := r(\vdash x \dashv)$ is its transformation under r, then clearly $|y| \leq \mu(|x|)$ and the two cases of interest are as follows:

- If $x \notin L(A)$, then $y \notin \mathrm{A}_{\mathrm{2NFA}}$, hence M does not accept y, so $\tilde{B}$ does not accept y either, causing B not to accept x.
- If $x \in L(A)$ and $|x| \leq m$, then $y \in \mathrm{A}_{\mathrm{2NFA}}$, hence M accepts y, hence $\tilde{B}$ also accepts y (since $|y| \leq \mu(m)$), causing B to accept x.

Overall, B may disagree with A only on $x \in L(A)$ with $|x| > m$, as promised.

[(B)⇒(A)] Suppose the logarithmic-space transducer T of the statement exists. Then $\mathrm{A}_{\mathrm{2NFA}}$ reduces to $\mathrm{A}_{\mathrm{2DFA}}$ in logarithmic space, as follows. Given a 2NFA A

and an input x, we apply the following two logarithmic-space transductions, where $m = |x|$:

$$\langle A\rangle\#\langle x\rangle \longrightarrow \langle A\rangle\#0^m\#\langle x\rangle \xrightarrow{T} \langle B\rangle\#\langle x\rangle.$$

First, we insert $\#0^m$ by running an obvious logarithmic-space transducer that performs the algebra for $m = |\langle x\rangle|/\lg\sigma$, where σ is the number of symbols in A. Then, we run T on $\langle A\rangle\#0^m$ to replace it with the encoding of a 2DFA B that agrees with A on all inputs of length $\leq m$, including x. Clearly, $\langle A\rangle\#\langle x\rangle \in \mathrm{A}_{\mathrm{2NFA}}$ iff $\langle B\rangle\#\langle x\rangle \in \mathrm{A}_{\mathrm{2DFA}}$.

Now, the NL-hardness of $\mathrm{A}_{\mathrm{2NFA}}$, the logarithmic-space reduction to $\mathrm{A}_{\mathrm{2DFA}}$, the fact that $\mathrm{A}_{\mathrm{2DFA}} \in \mathsf{L}$ (easy, just simulate the 2DFA, being careful not to loop), and the closure of L under logarithmic-space reductions imply $\mathsf{L} \supseteq \mathsf{NL}$. □

It is now clear that, when we restrict to inputs of length at most $m = \mathrm{poly}(s)$, the bound for the states in B is $\mathrm{poly}(s\sigma)$—not $\mathrm{poly}(s)$. Hence, for a 2NFA A over an alphabet of size $\sigma = 2^{\Omega(s)}$ (e.g., a 2NFA for TWL_s), the bound of Theorem 3.0 for the number of states in a 2DFA simulating A on inputs of length $\mathrm{poly}(s)$ cannot be lower than $2^{O(s)}$. We stress that this looseness is inherent in the Berman-Lingas argument, not just in our analysis of it: a larger alphabet for A implies a longer encoding $\langle A\rangle$, thus longer inputs to the alleged logarithmic-space DTM M for $\mathrm{A}_{\mathrm{2NFA}}$, more work space for M when it computes on these inputs, more internal configurations, more states in the 2DFA $\tilde{B}$ simulating M, and thus more states in the final 2DFA B simulating $\tilde{B}$.

It is also clear that the theorem's equivalence requires that the relationship between the given A and m and the promised B is constructive. That is, if (B) were modified to state that for every 2NFA A and length bound m a fitting 2DFA B simply *exists* (as opposed to it being logarithmic-space constructible), then the argument for (B)⇒(A) would not stand. This is an extra obstacle in connecting L versus NL with the purely existential question of 2D versus 2N (conceivably, a proof that $\mathsf{2D} \supseteq \mathsf{2N}$ needs no logarithmic-space conversion of s-state 2NFAs to $\mathrm{poly}(s)$-state 2DFAs).

Our Theorem 3.1 is a variant of Theorem 3.0 that removes both dependences above, on alphabet size and logarithmic-space constructibility. To remove the dependence on alphabet size, we switch to another NL-complete problem; to remove the dependence on constructibility, we switch to non-uniform L. The structure of the argument, however, is the same. Note that, of the six equivalences established among the four statements, (A)⇔(B) is the one closest to the statement of Theorem 3.0; (C)⇔(D) is the equivalence claimed in Sect. 1.1 between Questions (3) and (3′); and (D)⇔(A) is the statement in terms of complexity classes, as announced in Sect. 1.3.

Theorem 3.1 *The following statements are equivalent*:

(A) $\mathsf{L/poly} \supseteq \mathsf{NL}$.

(B) *There exists a polynomial q such that for every* 2NFA *A and length bound m there exists a* 2DFA *B that has at most $q(sm)$ states, for s the number of states in A, and agrees with A on all inputs of length at most m.*

(C) *There exists a polynomial p such that every s-state* 2NFA *has a* 2DFA *with $\leq p(s)$ states that agrees with it on all inputs of length $\leq s$.*

(D) $\mathsf{2D} \supseteq \mathsf{2N/poly}$.

Proof

[(A)⇒(B)] Suppose L/poly ⊇ NL. Then TWL JOIN ∈ L/poly. Therefore, some DTM M under strong advice $\mathscr{y}$ of length $g(m) = \text{poly}(m)$ solves TWL JOIN in space strongly bounded by $f(m) = \log m$.
Pick any (s,σ)-2NFA A and length bound m. We know that $L(A) \leq_{\text{h}}$ TWL JOIN via a reduction of expansion $\mu(m) = O(s^2 m)$ (Lemma 2.5). Hence, finding a 2DFA B for $L(A)$ and inputs of length $\leq m$ reduces to finding a 2DFA $\tilde{B}$ for $L(M,\mathscr{y})$ and inputs of length $\leq \mu(m)$ (Lemma 2.3). Indeed, such a $\tilde{B}$ is guaranteed by Lemma 2.2, with a number of states at most exponential in $f(\mu(m)) + \log g(\mu(m))$. Calculating

$$f\big(\mu(m)\big) = \log O\big(s^2 m\big) = O\big(\log(sm)\big),$$
$$\log g\big(\mu(m)\big) = \log \text{poly}\big(O\big(s^2 m\big)\big) = \log \text{poly}(sm) = O\big(\log(sm)\big),$$
$$2^{O(f(\mu(m))+\log g(\mu(m)))} = 2^{O(\log(sm))} = \text{poly}(sm),$$

we see that the number of states in B is $2 \cdot \text{poly}(sm) = \text{poly}(sm)$, as required.
[(B)⇒(C)] Applying (B) with $m = s$, we deduce (C) with $p(s) := q(s^2)$.
[(C)⇒(D)] Assume (C). Pick any family $\mathscr{L} = (\mathfrak{L}_h)_{h\geq 1} \in$ 2N/poly. Then every $\mathfrak{L}_h$ has instances of length $\leq q_1(h)$ and is solved by some 2NFA A_h with $\leq q_2(h)$ states, for q_1 and q_2 two polynomials. Let $q := q_1 + q_2$. Then clearly every $\mathfrak{L}_h$ has instances of length $\leq q(h)$ and is solved by some $q(h)$-state 2NFA A'_h. Now, (C) implies that every A'_h has a 2DFA B_h with $\leq p(q(h))$ states that agrees with it on all inputs of length $\leq q(h)$, and is thus correct on all instances of $\mathfrak{L}_h$. Hence, $\mathscr{B} = (B_h)_{h\geq 1}$ is a family of small 2DFAs that solve $\mathscr{L}$. Therefore, $\mathscr{L} \in$ 2D.
[(D)⇒(A)] Suppose 2D ⊇ 2N/poly. Pick any $L \in$ NL. Let Σ be its alphabet and let M be a NTM solving it (under empty advice) in space strongly bounded by $\log n$. For each $h \geq 1$, consider the problem $\mathfrak{L}_h$ that restricts L to inputs of length $< h$:

$$\mathfrak{L}_h := \big(\{x \in L \mid |x| < h\},\ \{x \in \overline{L} \mid |x| < h\}\big).$$

By Corollary 2.1(a) for $m = h-1$, we know some 2NFA A_h with $\text{poly}(h-1) = \text{poly}(h)$ states agrees with M on every input of length $< h$, and thus decides correctly on every instance of $\mathfrak{L}_h$. So, $\mathscr{L} := (\mathfrak{L}_h)_{h\geq 1}$ is a family of polynomially long problems, solvable by the family of small 2NFAs $(A_h)_{h\geq 1}$. Therefore $\mathscr{L}$ is in 2N/poly, and thus also in 2D (by our starting assumption). So, there exists a 2DFA family $\mathscr{B} = (B_h)_{h\geq 1}$ and a polynomial p such that every B_h solves $\mathfrak{L}_h$ with $\leq p(h)$ states.
Now consider the binary encodings of the automata of $\mathscr{B}$ as advice strings,

$$\mathscr{y} = (y_m)_{m\geq 0} := \big(\langle B_{m+1}\rangle\big)_{m\geq 0}$$

and let U be the DTM which, on input $x \in \Sigma^*$ and advice $y \in \{0,1\}^*$, interprets y as the encoding of a 2DFA and simulates it on x. Then $L(U,\mathscr{y}) = L$, because

$$\begin{aligned} x \in L(U,\mathscr{y}) &\iff U \text{ accepts } x \text{ and } y_n \quad (\text{definition of } L(U,\mathscr{y}), \text{ and } n := |x|) \\ &\iff B_{n+1} \text{ accepts } x \quad (\text{definition of } y_n \text{ and of } U) \\ &\iff x \in L \quad (B_{n+1} \text{ solves } \mathfrak{L}_{n+1}, \text{ and } |x| < n+1). \end{aligned}$$

In addition, $\mathscr{y}$ is *strong* advice for U, because every y_m helps U treat an input x 'correctly' (namely, as it would under $y_{|x|}$) not only when $|x| = m$ but also when $|x| < m$: Indeed, for all m and all x with $|x| = n \leq m$,

$$\begin{aligned} U \text{ accepts } x \text{ and } y_m &\iff B_{m+1} \text{ accepts } x \quad (\text{definition of } y_m \text{ and of } U) \\ &\iff x \in L \quad (B_{m+1} \text{ solves } \mathfrak{L}_{m+1}, \text{ and } |x| < m+1) \\ &\iff B_{n+1} \text{ accepts } x \quad (B_{n+1} \text{ solves } \mathfrak{L}_{n+1}, \text{ and } |x| < n+1) \\ &\iff U \text{ accepts } x \text{ and } y_n \quad (\text{definition of } y_n \text{ and of } U). \end{aligned}$$

Moreover, $|y_m| = |\langle B_{m+1}\rangle| = O(p(m+1)^2 \cdot |\Sigma|) = \text{poly}(m)$, since Σ is constant in m. Finally, the space used by U for x and y_m when $|x| \leq m$ is the space for storing the pointers into y_m that are necessary for the simulation of B_{m+1}, namely $O(\log |y_m|) = O(\log \text{poly}(m)) = O(\log m)$. Overall, U and $\mathscr{y}$ witness that $L \in \mathsf{L/poly}$. □

While Theorem 3.1 uncovers the full details of the Berman-Lingas connection between space-bounded TMs and size-bounded 2FAs, our next two theorems reveal that this connection is only one aspect of a deeper relationship. Specifically, they show that the equivalences of Theorem 3.1 remain valid when we appropriately decrease the space usage and advice length on the TM side and increase the input lengths on the 2FA side. Theorem 3.2 is the case where the space usage and the advice length decrease respectively to sub-logarithmic and sub-linear, whereas the input length increases to quasi-polynomial. Theorem 3.3 is the case of log-logarithmic space usage, poly-logarithmic advice, and exponential input lengths.

Because Theorem 3.3 is easier to compare to Theorem 3.1, we will state and prove it before Theorem 3.2. Both proofs use arguments very similar to those used for Theorem 3.1, so we will present them in less detail. The most important difference is that TWL JOIN is replaced respectively by TWL LONG-JOIN and by TWL JOIN_k when we prove the implications (A)⇒(B). The remaining differences are in the calculations.

Theorem 3.3 *The following statements are equivalent*:

(A) $\mathsf{LL/polylog} \supseteq \mathsf{NLL}$.
(B) *There exists a polynomial q such that for every* 2NFA *A and length bound m there exists a* 2DFA *B that has at most $q(s \log m)$ states, for s the number of states in A, and agrees with A on all inputs of length at most m.*
(C) *There exists a polynomial p such that every s-state* 2NFA *has a* 2DFA *with $\leq p(s)$ states that agrees with it on all inputs of length $\leq 2^s$.*
(D) $\mathsf{2D} \supseteq \mathsf{2N/exp}$.

Proof

[(A)⇒(B)] If $\mathsf{LL/polylog} \supseteq \mathsf{NLL}$ then TWL LONG-JOIN is in $\mathsf{LL/polylog}$, so it is solved by a DTM M under strong advice $\mathscr{y}$ of length $g(m) = \text{poly}(\log m)$ and strong space bound $f(m) = \log \log m$. As in Theorem 3.1, for every (s, σ)-2NFA A and length bound m, we know $L(A) \leq_{\text{h}} \text{TWL LONG-JOIN}$ via a reduction of expansion $\mu(m) = 2^{O(s)}m$ (Lemma 2.5), so it suffices to find a 2DFA $\tilde{B}$ for $L(M, \mathscr{y})$

on lengths $\leq \mu(m)$ (Lemma 2.3), which indeed exists with $2^{O(f(\mu(m))+\log g(\mu(m)))}$ states (Lemma 2.2). Now,

$$f\big(\mu(m)\big) = \log\log\big(2^{O(s)}m\big) = O\big(\log(s+\log m)\big),$$
$$\log g\big(\mu(m)\big) = \log \mathrm{poly}\big(\log\big(2^{O(s)}m\big)\big) = O\big(\log\log\big(2^{O(s)}m\big)\big)$$
$$= O\big(\log(s+\log m)\big),$$
$$2^{O(f(\mu(m))+\log g(\mu(m)))} = 2^{O(\log(s+\log m))} = \mathrm{poly}(s+\log m) = \mathrm{poly}(s\log m),$$

so our 2DFA for $L(A)$ on lengths $\leq m$ has $2\cdot\mathrm{poly}(s\log m) = \mathrm{poly}(s\log m)$ states.

[(B)⇒(C)] Applying (B) with $m = 2^s$, we deduce (C) with $p(s) := q(s^2)$.

[(C)⇒(D)] Assume (C), and let $\mathscr{L} = (\mathfrak{L}_h)_{h\geq 1} \in \mathsf{2N/exp}$. Then every $\mathfrak{L}_h$ has instances of length $\leq 2^{q_1(h)}$ and is solved by a 2NFA A_h with $\leq q_2(h)$ states, for q_1, q_2 two polynomials. So, $q := q_1 + q_2$ is such that every $\mathfrak{L}_h$ has instances of length $\leq 2^{q(h)}$ and is solved by a $q(h)$-state 2NFA A'_h. By (C), every A'_h has a 2DFA B_h with $\leq p(q(h))$ states that agrees with it on all inputs of length $\leq 2^{q(h)}$, including the instances of $\mathfrak{L}_h$. So, the small automata $\mathscr{B} = (B_h)_{h\geq 1}$ solve $\mathscr{L}$, causing $\mathscr{L} \in \mathsf{2D}$.

[(D)⇒(A)] Suppose $\mathsf{2D} \supseteq \mathsf{2N/exp}$. Pick any $L \in \mathsf{NLL}$. Let Σ be its alphabet, and M a NTM solving L (under empty advice) in space strongly bounded by $\log\log n$. As in Theorem 3.1, consider for each $h \geq 1$ the problem $\mathfrak{L}_h$ that restricts L to instances of bounded length, this time exponential in h:

$$\mathfrak{L}_h := \big(\big\{x \in L \mid |x| < 2^{h-1}\big\},\ \big\{x \in \overline{L} \mid |x| < 2^{h-1}\big\}\big).$$

By Corollary 2.1(b) for $m = 2^{h-1}-1$, we know some 2NFA A_h with $\mathrm{poly}(\log(2^{h-1}-1)) = \mathrm{poly}(h)$ states agrees with M on every input of length $< 2^{h-1}$, and thus decides correctly on every instance of $\mathfrak{L}_h$. So, the family $\mathscr{L} := (\mathfrak{L}_h)_{h\geq 1}$ of exponentially long problems is solved by the family $(A_h)_{h\geq 1}$ of small 2NFAs. Hence, $\mathscr{L} \in \mathsf{2N/exp}$, and thus also $\mathscr{L} \in \mathsf{2D}$ (by our starting assumption). So, some 2DFA family $\mathscr{B} = (B_h)_{h\geq 1}$ and polynomial p are such that every B_h solves $\mathfrak{L}_h$ with $p(h)$ states.

We now continue with the same DTM U as in the proof of Theorem 3.1, advised by the encodings of the 2DFAs of $\mathscr{B}$, this time 'spread out' exponentially:

$$\mathscr{Y} = (y_m)_{m\geq 0} := \big(\langle B_{h(m)}\rangle\big)_{m\geq 0} \quad \text{where } h(m) := \lceil \log(m+1)\rceil + 1.$$

So, U under $\mathscr{Y}$ decides whether to accept an input x by simulating $B_{h(|x|)}$. Since this 2DFA solves $\mathfrak{L}_{h(|x|)}$, it decides L correctly on all inputs strictly shorter than

$$2^{h(|x|)-1} = 2^{\lceil \log(|x|+1)\rceil} \geq 2^{\log(|x|+1)} = |x|+1,$$

and thus also on x. Therefore, U under $\mathscr{Y}$ decides L correctly on all inputs, namely $L(U,\mathscr{Y}) = L$. It is also easy to verify that every $B_{h(m)}$ helps U decide correctly not only on all inputs of length exactly m but also on all inputs of length $< m$, namely $\mathscr{Y}$ is *strong* advice for U. Finally,

$$|y_m| = \big|\langle B_{h(m)}\rangle\big| = O\big(p\big(h(m)\big)^2 \cdot |\Sigma|\big) = O\big(p\big(h(m)\big)^2\big) = \mathrm{poly}(\log m)$$

and the space used by U for the pointers into y_m that are necessary to simulate $B_{h(m)}$ is $O(\log|y_m|) = O(\log \mathrm{poly}(\log m)) = O(\log\log m)$. Overall, $L \in \mathsf{LL/polylog}$. □

Theorem 3.2 *For any $k \geq 1$, the following statements are equivalent*:

(A) $\mathsf{DSPACE}(\sqrt[k]{\log n})/2^{O(\sqrt[k]{\log n})} \supseteq \mathsf{NSPACE}(\sqrt[k]{\log n})$.

(B) *There exists a polynomial q such that for every* 2NFA *A and length bound m there exists a* 2DFA *B that has at most $q(s\,2^{\sqrt[k]{\log m}})$ states, for s the number of states in A, and agrees with A on all inputs of length at most m.*

(C) *There exists a polynomial p such that every s-state* 2NFA *has a* 2DFA *with $\leq p(s)$ states that agrees with it on all inputs of length $\leq 2^{\log^k s}$.*

(D) $\mathsf{2D} \supseteq \mathsf{2N}/2^{O(\log^k h)}$.

Proof

[(A)⇒(B)] If (A) holds, then a DTM M under strong advice of length $2^{O(\sqrt[k]{\log m})}$ solves TWL JOIN$_k$ in space strongly bounded by $\sqrt[k]{\log m}$. Given a (s,σ)-2NFA A and length bound m, we now have a reduction of $L(A)$ to TWL JOIN$_k$ with expansion $\mu(m) = 2^{O(\log^k s)}m$, and a 2DFA $\tilde{B}$ that solves TWL JOIN$_k$ on lengths $\leq \mu(m)$ with a number of states at most exponential in

$$\sqrt[k]{\log \mu(m)} + \log 2^{O(\sqrt[k]{\log \mu(m)})} = O\Big(\sqrt[k]{\log \mu(m)}\Big) = O\left(\sqrt[k]{\log^k s + \log m}\right)$$
$$= O\left(\sqrt[k]{\log^k s} + \sqrt[k]{\log m}\right) = O\Big(\log s + \sqrt[k]{\log m}\Big).$$

So, some 2DFA B solves $L(A)$ on lengths $\leq m$ with a number of states at most

$$2 \cdot 2^{O(\log s + \sqrt[k]{\log m})} = 2 \cdot \mathrm{poly}\Big(2^{\log s + \sqrt[k]{\log m}}\Big) = \mathrm{poly}\Big(s\,2^{\sqrt[k]{\log m}}\Big).$$

[(B)⇒(C)] Applying (B) with $m = 2^{\log^k s}$, we deduce (C) with $p(s) := q(s^2)$.

[(C)⇒(D)] Assuming (C), we let $\mathscr{L} = (\mathfrak{L}_h)_{h\geq 1} \in \mathsf{2N}/2^{O(\log^k h)}$ and pick q such that every $\mathfrak{L}_h$ has instances of length $\leq 2^{\log^k q(h)}$ and is solved by a $q(h)$-state 2NFA A'_h. By (C), every A'_h has a 2DFA B_h with $\leq p(q(h))$ states that agrees with it on all inputs of length $\leq 2^{\log^k q(h)}$, and thus solves $\mathfrak{L}_h$. Hence $\mathscr{L} \in \mathsf{2D}$.

[(D)⇒(A)] Suppose $\mathsf{2D} \supseteq \mathsf{2N}/2^{O(\log^k h)}$ and some NTM solves L under empty advice in space strongly bounded by $\sqrt[k]{\log n}$. For every $h \geq 1$, we consider the restriction $\mathfrak{L}_h$ of L to instances of length $< 2^{\lceil \log h\rceil^k}$. Applying Lemma 2.2 for $m = 2^{\lceil\log h\rceil^k} - 1$ and $f(m) = \sqrt[k]{\log m}$ and $g(m) = 0$, we see that $\mathfrak{L}_h$ is solved by a 2NFA with a number of states exponential in

$$f(m) = \sqrt[k]{\log m} = \sqrt[k]{\log\big(2^{\lceil\log h\rceil^k} - 1\big)} = O(\log h)$$

and thus polynomial in h. Hence, $(\mathfrak{L}_h)_{h\geq 1}$ is in $\mathsf{2N}/2^{O(\log^k h)} \subseteq \mathsf{2D}$, and thus solvable by a family $(B_h)_{h\geq 1}$ of small 2DFAs. Using the encodings of these 2DFAs as

advice

$$\mathscr{y} = (y_m)_{m\geq 0} := \big(\langle B_{h(m)}\rangle\big)_{m\geq 0} \quad \text{where } h(m) := \lceil 2^{\sqrt[k]{\log(m+1)}} \rceil$$

to the DTM U from the proof of Theorem 3.1, we again verify that $L(U, \mathscr{y}) = L$, that $\mathscr{y}$ is strong advice of length $2^{O(\sqrt[k]{\log m})}$, and that the space usage is $O(\sqrt[k]{\log m})$. □

4 Conclusion

We strengthened and deepened an old theorem of Berman and Lingas (Theorem 3.0), which connected the comparison between determinism and nondeterminism in size-bounded two-way finite automata to the comparison between determinism and nondeterminism in space-bounded Turing machines. We proved three variants of that theorem, one which elaborated directly on it and tightened its statement (Theorem 3.1), and two more which extended this tightened statement to other combinations of bounds for the relevant resources (Theorems 3.2 and 3.3).

Our arguments suggest a more general proof scheme, applicable not only in the comparison between determinism and nondeterminism, but also in the comparison of other modes of computation. As evidence for this generality, we stated without proof a variant of the Berman-Lingas Theorem for the comparison between determinism and alternation (Theorem 1.4). We expect that several theorems of this kind will strengthen the connection between the size complexity of two-way finite automata and the space complexity of Turing machines, for the benefit of both theories.

In light of our arguments, it would also be interesting to explore possible improvements of the implication (7), the recent theorem of Geffert and Pighizzini from [4]. It is not hard to see that this can be strengthened by replacing its assumption that $\mathsf{L} \supseteq \mathsf{NL}$ with the weaker assumption that $\mathsf{L/poly} \supseteq \mathsf{NL}$. It is then likely that the converse becomes true as well, but proving it appears not to be as straightforward.

References

1. Alberts, M.: Space complexity of alternating Turing machines. In: Proceedings of FCT, pp. 1–7 (1985)
2. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Proceedings of FOCS, pp. 218–223 (1979)
3. Berman, P., Lingas, A.: On complexity of regular languages in terms of finite automata. Report 304, Institute of Computer Science, Polish Academy of Sciences, Warsaw (1977)
4. Geffert, V., Pighizzini, G.: Two-way unary automata versus logarithmic space. Inf. Comput. **209**(7), 1016–1025 (2011)
5. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theor. Comput. Sci. **295**, 189–203 (2003)
6. Hopcroft, J.E., Ullman, J.D.: Some results on tape-bounded Turing machines. J. ACM **16**(1), 168–177 (1967)
7. Hromkovič, J., Schnitger, G.: Nondeterminism versus determinism for two-way finite automata: generalizations of Sipser's separation. In: Proceedings of ICALP, pp. 439–451 (2003)
8. Ibarra, O.H., Ravikumar, B.: Sublogarithmic-space Turing machines, nonuniform space complexity, and closure properties. Math. Syst. Theory **21**, 1–17 (1988)

9. Kapoutsis, C.: Deterministic moles cannot solve liveness. J. Autom. Lang. Comb. **12**(1–2), 215–235 (2007)
10. Kapoutsis, C.: Nondeterminism is essential in small 2FAs with few reversals. In: Proceedings of ICALP, vol. 2, pp. 192–209 (2011)
11. Karp, R.M., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: Proceedings of STOC, pp. 302–309 (1980)
12. Kuroda, S.: Classes of languages and linear-bounded automata. Inf. Control **7**, 207–223 (1964)
13. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proceedings of STOC, pp. 275–286 (1978)
14. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. Syst. Sci. **4**, 177–192 (1970)
15. Seiferas, J.I.: Untitled manuscript, communicated to M. Sipser (1973)
16. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. IBM J. Res. Dev. **3**, 198–200 (1959)
17. Stearns, R.E., Hartmanis, J., Lewis, P.M. II: Hierarchies of memory limited computations. In: Proceedings of SWCT, pp. 179–190 (1965)
18. Szepietowski, A.: If deterministic and nondeterministic space complexities are equal for $\log\log n$ then they are also equal for $\log n$. In: Proceedings of STACS, pp. 251–255 (1989)