Theory of Computing Systems

Two-way automata characterizations of L/poly versus NL

--Manuscript Draft--

Manuscript Number:	
Full Title:	Two-way automata characterizations of L/poly versus NL
Article Type:	Special Issue: CSR 2012 - invited only
Keywords:	two-way finite automata; logarithmic space; structural complexity; descriptional complexity
Corresponding Author:	Giovanni Pighizzini, Ph.D. Universita' degli Studi Milano, ITALY
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Universita' degli Studi
Corresponding Author's Secondary Institution:	
First Author:	Christos Kapoutsis, Ph.D.
First Author Secondary Information:	
Order of Authors:	Christos Kapoutsis, Ph.D.
	Giovanni Pighizzini, Ph.D.
Order of Authors Secondary Information:	

Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Two-way automata characterizations of L/poly versus NL

Christos A. Kapoutsis · Giovanni Pighizzini

Received: date / Accepted: date

Abstract Let L/poly and NL be the standard complexity classes, of languages recognizable in logarithmic space by Turing machines which are deterministic with polynomially-long advice and nondeterministic without advice, respectively. We recast the question whether L/poly $\supseteq$ NL in terms of deterministic and nondeterministic two-way finite automata (2DFAs and 2NFAs). We prove it equivalent to the question whether every s-state *unary* 2NFA has an equivalent poly(s)-state 2DFA, or whether a poly(h)-state 2DFA can check accessibility in h-vertex graphs (even under unary encoding) or check two-way liveness in h-tall, h-column graphs. This complements two recent improvements of an old theorem of Berman and Lingas. On the way, we introduce new types of reductions between regular languages (even unary ones), use them to prove the completeness of specific languages for two-way nondeterministic polynomial size, and propose a purely combinatorial conjecture that implies L/poly $\not\supseteq$ NL.

Keywords Two-way finite automata · Logarithmic space · Structural complexity · Descriptional complexity

1 Introduction

A prominent open question in complexity theory asks whether nondeterminism is essential in logarithmic-space Turing machines. Formally, this is the question

Preliminary version presented in the 7th International Computer Science Symposium in Russia, Nizhny Novgorod, July 3-7, 2012 [Lecture Notes in Computer Science vol. 7353, Springer-Verlag, pp. 217-228].

C.A. Kapoutsis
LIAFA, Université Paris Diderot - Paris VII - Case 7014, F-75205 Paris Cedex 13.
E-mail: christos.kapoutsis@liafa.univ-paris-diderot.fr

G. Pighizzini
DI, Università degli Studi di Milano, I-20135 Milano.
E-mail: pighizzini@di.unimi.it

whether $\mathsf{L} = \mathsf{NL}$, for L and NL the standard classes of languages recognizable by logarithmic-space deterministic and nondeterministic Turing machines.

In the late 70's, Berman and Lingas [1] connected this question to the comparison between deterministic and nondeterministic two-way finite automata (2DFAs and 2NFAs), proving that *if* $\mathsf{L} = \mathsf{NL}$*, then for every* s*-state* σ*-symbol* 2NFA *there is a* poly$(s\sigma)$*-state* 2DFA *which agrees with it on all inputs of length* $\leq s$. (They also proved that this implication becomes an equivalence if we require that the 2DFA be constructible from the 2NFA in logarithmic space.)

Recently, two improvements of this theorem have been announced. On the one hand, Geffert and Pighizzini [5] proved that *if* $\mathsf{L} = \mathsf{NL}$ *then for every* s*-state unary* 2NFA *there is an equivalent* poly(s)*-state* 2DFA. That is, in the special case where $\sigma = 1$, the Berman-Lingas theorem becomes true without any restriction on input lengths. On the other hand, Kapoutsis [8] proved that $\mathsf{L/poly} \supseteq \mathsf{NL}$ *iff for every* s*-state* σ*-symbol* 2NFA *there is a* poly(s)*-state* 2DFA *which agrees with it on all inputs of length* $\leq s$, where $\mathsf{L/poly}$ is the standard class of languages recognizable by deterministic logarithmic-space Turing machines with polynomially-long advice. Hence, the Berman-Lingas theorem is true even under the weaker assumption $\mathsf{L/poly} \supseteq \mathsf{NL}$, even for the stronger conclusion where the 2DFA size is independent of σ, and then even in the converse direction. He also proved variants of this equivalence for other combinations of bounds for the space usage, the advice length, and the length of the inputs.

A natural question arising from these developments is whether the theorem of [5] can be strengthened to resemble that of [8]: *Does the implication remain true under the weaker assumption that* $\mathsf{L/poly} \supseteq \mathsf{NL}$*? If so, does it then become an equivalence?* Indeed, we prove that $\mathsf{L/poly} \supseteq \mathsf{NL}$ *iff for every* s*-state unary* 2NFA *there is an equivalent* poly(s)*-state* 2DFA. Intuitively, this means that $\mathsf{L/poly} \supseteq \mathsf{NL}$ is true not only iff 'small' 2NFAs can be simulated by 'small' 2DFAs on *'short' inputs* (as in [8]), but also iff the same is true for *unary inputs*.

In this light, a second natural question is whether this common behavior of 'short' and unary inputs is accidental or follows from deeper common properties. Indeed, our analysis reveals two such properties. They are related to *outer-nondeterministic* 2FAs (2OFAs, which perform nondeterministic choices only on the end-markers [2]) and to the *graph accessibility problem* (GAP, the problem of checking the existence of paths in directed graphs), and use the fact that checking whether a 'small' 2OFA M accepts an input x reduces to solving GAP in a 'small' graph $G_M(x)$.

- The first common property is that, both on 'short' and on unary inputs, 'small' 2NFAs can be simulated by 'small' 2OFAs (Lemma 1).
- The second common property is that, both on 'short' and on unary inputs, it is possible to encode instances of GAP so that a 'small' 2DFA can extract $G_M(x)$ from x (Lemma 5) and simultaneously simulate on it another 'small' 2DFA (Lemma 6).

For 'short' inputs, both properties follow from standard ideas; for unary inputs, they follow from the analyses of [3,9] and a special unary encoding for graphs.

We work in the complexity-theoretic framework of [10]. We focus on the class 2D of (families of promise) problems solvable by polynomial-size 2DFAs, and on the corresponding classes 2N/poly and 2N/unary for 2NFAs and for problems with polynomially-long and with unary instances, respectively. In these terms, $\mathsf{2D} \supseteq \mathsf{2N/poly}$ means 'small' 2DFAs can simulate 'small' 2NFAs on *'short' inputs*; $\mathsf{2D} \supseteq \mathsf{2N/unary}$ means the same for *unary inputs*; the theorem of [8] is that $\mathsf{L/poly} \supseteq \mathsf{NL} \Leftrightarrow \mathsf{2D} \supseteq \mathsf{2N/poly}$; the theorem of [5] is the forward implication that $\mathsf{L} \supseteq \mathsf{NL} \Rightarrow \mathsf{2D} \supseteq \mathsf{2N/unary}$; and our main contribution is the stronger statement

$$\mathsf{L/poly} \supseteq \mathsf{NL} \iff \mathsf{2D} \supseteq \mathsf{2N/unary}\,. \tag{1}$$

This we derive from [8] and the equivalence $\mathsf{2D} \supseteq \mathsf{2N/poly} \Leftrightarrow \mathsf{2D} \supseteq \mathsf{2N/unary}$, obtained by our analysis of the common properties of 'short' and unary inputs.

Our approach returns several by-products of independent interest, already anticipated in [7] for enriching the framework of [10]: new types of reductions, based on 'small' two-way deterministic finite transducers; the completeness of binary and unary versions of GAP (BGAP and UGAP) for 2N/poly and 2N/unary, respectively, under such reductions (Corollary 2); the closure of 2D under such reductions (Corollary 3); and the realization of the central role of 2OFAs in this framework (as also recognized in [2]). In the end, our main theorem (Theorem 1) is the equivalence of $\mathsf{L/poly} \supseteq \mathsf{NL}$ to all these statements (and one more):

$$\mathsf{2D} \supseteq \mathsf{2N/poly} \quad \mathsf{2D} \supseteq \mathsf{2N/unary} \quad \mathsf{2D} \supseteq \mathsf{2O} \quad \mathsf{2D} \ni \text{BGAP} \quad \mathsf{2D} \ni \text{UGAP}$$

where 2O is the analogue of 2D for 2OFAs. Hence, the conjecture $\mathsf{L/poly} \not\supseteq \mathsf{NL}$ is now the conjecture that all these statements are false. In this context, we also propose a stronger conjecture, both in algorithmic and in combinatorial terms.

Concluding this introduction, we note that, of course, (1) can also be derived by a direct proof of each direction. In such a derivation, the forward implication is a straightforward strengthening of the proof of [5], but the backward implication needs the encoding of UGAP and the ideas behind Lemma 6.2.

2 Preparation

If x is a string, then $|x|$, x_i, and x^i are its length, its i-th symbol ($1 \le i \le |x|$), and the concatenation of i copies of it ($i \ge 0$). The empty string is denoted by ϵ.

If $h \ge 1$, we let $[h]{:=}\{0, 1, \ldots, h-1\}$, use K_h for the complete directed graph with vertex set $[h]$, and p_h for the h-th smallest prime number. Given a subgraph G of K_h, we consider the following two encodings. (See Fig. 1 for an example.)

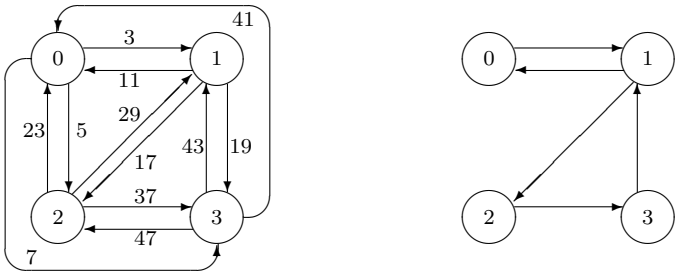

Fig. 1 The complete graph K_4 with a subgraph G. Each edge (i, j) of K_4 is marked with the $(i{\cdot}h{+}j{+}1)$-th prime number (for simplicity, self-loops are not shown in the picture). The encodings of G are $\langle G\rangle_2 = \mathtt{0100101000010100}$ and $\langle G\rangle_1 = \mathtt{0}^{3\cdot 11\cdot 17\cdot 37\cdot 43} = \mathtt{0}^{892551}$.

- The *binary encoding* of G, denoted as $\langle G\rangle_2$, is the standard h^2-bit encoding of the adjacency matrix of G: arrow (i, j) is present in G iff the $(i{\cdot}h{+}j{+}1)$-th most significant bit of the encoding is `1`.
- The *unary encoding* of G, denoted as $\langle G\rangle_1$, uses the natural correspondence between the bits of $\langle G\rangle_2$ and the h^2 smallest prime numbers, where the k-th most significant bit maps to p_k, and thus arrow (i, j) maps to the prime number $p_{(i,j)} := p_{i\cdot h+j+1}$.
  We let $\langle G\rangle_1 := \mathtt{0}^{n_G}$, where the length n_G is the product of the primes which correspond to the `1`s of $\langle G\rangle_2$, and thus to the arrows of K_h which are present in G:
  $$n_G := \prod_{\text{bit } k \text{ of } \langle G\rangle_2 \text{ is } \mathtt{1}} p_k \quad = \prod_{(i,j) \text{ is in } G} p_{(i,j)} \quad = \prod_{(i,j) \text{ is in } G} p_{i\cdot h+j+1} \ . \tag{2}$$
  Note that, conversely, every length $n \geq 1$ determines the unique subgraph $K_h(n)$ of K_h where each arrow (i, j) is present iff the corresponding prime $p_{(i,j)}$ divides n. Easily, $G = K_h(n_G)$.

A *prime encoding* of a length $n \geq 1$ is any string $\#z_1\#z_2\#\cdots\#z_m \in (\#\Sigma^*)^*$, where $\# \notin \Sigma$ and each z_i encodes one of the m prime powers in the prime factorization of n. Here, the alphabet Σ and the encoding scheme for the z_i are arbitrary, but fixed; the order of the z_i is arbitrary.

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 $x \in L \cup \tilde{L}$. If $\tilde{L} = \Sigma^* - L$ then $\mathfrak{L}$ is a *language*. If $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1}$ is a family of problems, its members are *short* if every instance of every $\mathfrak{L}_h$ has length $\leq p(h)$, for some polynomial p. If $\mathcal{M} = (M_h)_{h\geq 1}$ is a family of machines, its members are *small* if every M_h has $\leq p(h)$ states, for some polynomial p.

2.1 Automata

A *two-way nondeterministic finite automaton* (2NFA) consists of a finite control and an end-marked, read-only tape, accessed via a two-way head. Formally, it

is a tuple $M = (S, \Sigma, \delta, q_0, q_f)$ of a set of states S, an alphabet Σ, a start state $q_0 \in S$, a final state $q_f \in S$, and a set of transitions

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

for $\vdash,\dashv \notin \Sigma$ two end-markers and L:=−1 and R:=+1 the two directions. An input $x \in \Sigma^*$ is presented on the tape as $\vdash x \dashv$. The computation starts at q_0 on $\vdash$. At each step, M uses δ on the current state and symbol to decide the possible next state-move pairs. We assume δ never violates an end-marker, except if on $\dashv$ and the next state is q_f. A *configuration* is a state-position pair in $S \times [\,|x|+2\,]$ or the pair $(q_f, |x|+2)$. The computation produces a tree of configurations, and x is *accepted* if some branch 'falls off $\dashv$ into q_f', i.e., ends in $(q_f, |x|+2)$. A problem $(L, \tilde{L})$ is *solved* by M if M accepts all $x \in L$ but no $x \in \tilde{L}$.

We consider some restricted classes of 2NFAs.

- M is *outer-nondeterministic* (2OFA [2]) if all nondeterministic choices are made on the end-markers: formally, for each $(q, a) \in S \times \Sigma$ there is at most one transition of the form $(q, a, \,.\,, \,.\,)$.
- M is *deterministic* (2DFA) if the previous condition holds also for each $(q, a) \in S \times \{\vdash,\dashv\}$.
- M is *sweeping* (SNFA, SOFA, SDFA) if the head reverses only on the end-markers: formally, the set of transitions is now just

  $$\delta \subseteq S \times (\Sigma \cup \{\vdash,\dashv\}) \times S$$

  and the next position is defined to be always the adjacent one in the direction of motion; except if the head is on $\vdash$ or if the head is on $\dashv$ and the next state is not q_f, in which two cases the head reverses.
- M is *rotating* (RNFA, ROFA, RDFA) if it is sweeping, but modified so that its head jumps directly back to $\vdash$ every time it reads $\dashv$ and the next state is not q_f: formally, the next position is now always the adjacent one to the right; except when the head is on $\dashv$ and the next state is not q_f, in which case the next position is on $\vdash$. Restricting the general definition of outer-nondeterminism, we require that a ROFA makes all nondeterministic choices exclusively on $\vdash$.

Lemma 1 *For every s-state* 2NFA *M and length bound l, there is a $O(s^2 l^2)$-state* ROFA *M' that agrees with M on all inputs of length $\leq l$. If M is unary, then M' has $O(s^2)$ states and agrees with M on all inputs (of any length).*

Proof Pick any 2NFA $M = (S, \Sigma, \delta, q_0, q_f)$ and length bound l. To simulate M on inputs x of length $n \leq l$, a ROFA $M' = \big(S', \Sigma, \,.\,, (q_0, 0), q'_f\big)$ uses the states

$$S' := \big(S \times [l{+}2]\big) \cup \big(S \times [l{+}2] \times S \times \{\text{L,R}\} \times [l{+}1]\big) \cup \{\, q'_f \,\}\,.$$

The event of M being in state q and on tape cell i (where $0 \leq i \leq n{+}1$, cell 0 contains $x_0{:=}\vdash$ and cell $n{+}1$ contains $x_{n+1}{:=}\dashv$) is simulated by M' being in state (q, i) and on $\vdash$. From there, M' nondeterministically 'guesses' among

all $(p,d) \in S \times \{\text{L},\text{R}\}$ a 'correct' next transition of M out of q and x_i (i.e., a transition leading to acceptance) and goes on to verify that this transition is indeed valid, by deterministically sweeping the input up to cell i (using states of the form (q,i,p,d,j) with $j<i$) and checking that indeed $(q,x_i,p,d) \in \delta$. If the check fails, then M' just hangs (in this branch of the computation). Otherwise, it continues the sweep in state $(p,i+d)$ until it reaches $\dashv$ and jumps back to $\vdash$; except if $x_i = \dashv$ & $p = q_\text{f}$ & $d = \text{R}$, in which case it just falls off $\dashv$ into q'_f.[1]

If M is unary, then it can be simulated by a SOFA $\tilde{M}$ with $\tilde{s} = O(s^2)$ states [3]. Without changing the set of states, this can then be converted into a ROFA M' that simply replaces backward sweeps with forward ones—a straightforward simulation, which is now presented for the sake of completeness.

Suppose $\tilde{M} = (\tilde{S}, ., ., \tilde{q}_0, \tilde{q}_\text{f})$. By inspecting the construction in [3] we can observe that $\tilde{S}$ is partitioned in two sets $\tilde{S}_F, \tilde{S}_B$, such that:

- states in $\tilde{S}_F$ are used only in forward sweeps and never reached when the head is on $\vdash$,
- states in $\tilde{S}_B$ are used only in backward sweeps and never reached when the head is on $\dashv$,
- $\tilde{q}_0 \in \tilde{S}_B$ and $\tilde{q}_f \in \tilde{S}_F$.[2]

More precisely, with the head on $\vdash$ and in a state from $\tilde{S}_B$ (either the ending state of the previous backward sweep or $\tilde{q}_0$, in the case of the first sweep), a forward sweep starts by moving the head to the right and visiting only states in $\tilde{S}_F$, up until it reaches $\dashv$. There, $\tilde{M}$ may spawn a backward sweep, using states in $\tilde{S}_B$, up until it returns to $\vdash$. However, on $\dashv$ from some states it could be also possible to move right to accept in $\tilde{q}_\text{f}$.

Define $M' := (\tilde{S}, ., ., \tilde{q}_0, \tilde{q}_\text{f})$. Whenever in $q \in \tilde{S}_F$ and on $\mathtt{0}$, M' behaves exactly as $\tilde{M}$. Thus, forward sweeps are executed as in $\tilde{M}$. On reading $\dashv$, the simulation changes: if in $q \in \tilde{S}_F$ $\tilde{M}$ can violate the end-marker to accept in $\tilde{q}_\text{f}$ then M' deterministically makes the same move (hence, all other transitions from q on $\dashv$ are removed in M'); otherwise, M' simulates each backward sweep

[1] It is tempting to consider the following, simpler simulation. From (q,i) on $\vdash$, M' sweeps the tape deterministically up to cell i (using states of the form (q,i,j) with $j<i$), where it records x_i in its memory; it then completes the sweep in state (q,i,x_i), jumping back to $\vdash$ in that same state. From there, it transitions nondeterministically to states of the form $(p,i\pm1)$ according to the choices of M from q on x_i. This can be implemented with $O(sl^2+sl\sigma)$ states, where $\sigma = |\Sigma|$. If σ is constant in s, then this is $O(sl^2)$ states, better than in Lemma 1. If $\sigma = \text{poly}(s)$, then it increases to $\text{poly}(s)\cdot l^2$ states, still good for Corollary 1. But if σ is super-polynomial in s, then the size of M' becomes super-polynomial as well, and Corollary 1 cannot follow.

[2] Actually the construction in [3] assumes acceptance on $\vdash$ in $\tilde{q}_f$. Furthermore, $\tilde{q}_f$ can be entered only from some states in $\tilde{S}_B$ on $\vdash$, *without moving the input head.* To be consistent with the acceptance condition given in Section 2.1, we make the following minor changes to $\tilde{M}$, which do not affect the set of states. Each transition entering $\tilde{q}_f$ on $\vdash$ moves the head to the right. Furthermore, in the state $\tilde{q}_f$ $\tilde{M}$ always moves to the right, without changing state. This is done even on $\dashv$. In this way, once $\tilde{q}_f$ is entered, $\tilde{M}$ completely scans the input by remaining in $\tilde{q}_f$, in order to accept after violating $\dashv$.

of $\tilde{M}$ from q and $\dashv$ with a forward sweep from q and $\vdash$. Since reversing a unary input does not change it, this can be done in a natural way. In particular:[3]

- for any $q \in \tilde{S}_F$ such that $\tilde{M}$ does not have the transition $(q, \dashv, \tilde{q}_\mathrm{f})$, the transition $(q, \dashv, q)$ is added in M' (which causes M' to jump back to $\vdash$),[4]
- for any $q \in \tilde{S}_F$, $q' \in \tilde{S}_B$, every transition of the form $(q, \dashv, q')$ in $\tilde{M}$ (which would be moving to the left) is replaced by the transition $(q, \vdash, q')$ in M' (which moves right), and
- for any $q, q' \in \tilde{S}_B$, all the other transitions in $\tilde{M}$ of the form $(q, \mathtt{0}, q')$ (which would move left) are kept in M' (but now they move right).

If a backward sweep of $\tilde{M}$ ends on $\vdash$ in some $p \in \tilde{S}_B$, then the corresponding simulating forward sweep ends on $\dashv$ in the same state p. So, the head jumps back to $\vdash$ (since $p \neq \tilde{q}_\mathrm{f}$) and its configuration becomes identical to that of $\tilde{M}$ at the end of the backward sweep.

Clearly, M' behaves deterministically both on $\dashv$ (by construction) and on $\mathtt{0}$ (because $\tilde{M}$ does). So, all nondeterminism is on $\vdash$, and M' is indeed a ROFA. Its number of states is $\tilde{s}$. Hence, M' is also of size $O(s^2)$. □

A family of 2NFAs $\mathcal{M} = (M_h)_{h\geq 1}$ *solves* a family of problems $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1}$ if every M_h solves $\mathfrak{L}_h$. The class of families of problems that admit small 2NFAs is

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

Its restrictions to families of *short* and of *unary* problems are respectively $\mathsf{2N/poly}$ and $\mathsf{2N/unary}$. Analogous classes for restricted 2NFAs are named similarly: e.g., the class for problems with small 2DFAs is $\mathsf{2D}$, the class for short problems with small ROFAs is $\mathsf{RO/poly}$, etc.

Corollary 1 $\mathsf{2N/poly} = \mathsf{RO/poly}$ *and* $\mathsf{2N/unary} = \mathsf{RO/unary}$.

Proof The inclusions $\mathsf{2N/poly} \supseteq \mathsf{RO/poly}$ and $\mathsf{2N/unary} \supseteq \mathsf{RO/unary}$ follow from the definitions. The inclusions $\mathsf{2N/poly} \subseteq \mathsf{RO/poly}$ and $\mathsf{2N/unary} \subseteq \mathsf{RO/unary}$ are proved similarly, so we prove only the first one.

Pick any $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1} \in \mathsf{2N/poly}$. We know there exist polynomials p, q and a family of 2NFAs $\mathcal{M} = (M_h)_{h\geq 1}$ such that every $\mathfrak{L}_h$ is solved by M_h with $\leq p(h)$ states and has instances only of length $\leq q(h)$. By Lemma 1, for each M_h there is a ROFA M'_h with $O(p(h)^2 q(h)^2) = \mathrm{poly}(h)$ states which agrees with it on all inputs of length $\leq q(h)$, and thus solves $\mathfrak{L}_h$ as well. Therefore, $\mathcal{M}' := (M'_h)_{h\geq 1}$ is a family of small ROFAs which solves $\mathcal{L}$. Hence $\mathcal{L} \in \mathsf{RO}$, and thus $\mathcal{L} \in \mathsf{RO/poly}$ as well (since the problems are short). □

2.2 Graph accessibility

The *graph accessibility problem* on h vertices is:

[3] We remind the reader that, since $\tilde{M}$ is a SOFA, the direction of the head in its transitions is implicit, i.e, the set of transitions is a subset of $\tilde{S} \times (\Sigma \cup \{\vdash, \dashv\}) \times \tilde{S}$ (cf. p. 5).

[4] Notice that $\tilde{M}$ has the transition $(q_\mathrm{f}, \dashv, \tilde{q}_\mathrm{f})$ (cf. note 2). Hence, $q \neq q_\mathrm{f}$ here.

"Given a subgraph G of K_h, check that G contains a path from 0 to $h{-}1$."

Depending on whether G is given in binary or unary, we get the following two formal variants of the problem:

$$\begin{aligned}\textsc{bgap}_h :=\big(&\{\langle G\rangle_2 \mid G \text{ is subgraph of } K_h \text{ and has a path } 0 \rightsquigarrow h{-}1\},\\ &\{\langle G\rangle_2 \mid G \text{ is subgraph of } K_h \text{ and has no path } 0 \rightsquigarrow h{-}1\}\big)\\ \textsc{ugap}_h :=\big(&\{\mathtt{0}^n \mid K_h(n) \text{ has a path } 0 \rightsquigarrow h{-}1\},\\ &\{\mathtt{0}^n \mid K_h(n) \text{ has no path } 0 \rightsquigarrow h{-}1\}\big)\end{aligned}$$

and the corresponding families

$$\textsc{bgap} :=(\textsc{bgap}_h)_{h\geq 1} \text{ and } \textsc{ugap} := (\textsc{ugap}_h)_{h\geq 1}\,.$$

Lemma 2 $\textsc{bgap}_h$ *and* $\textsc{ugap}_h$ *are solved by* ROFAs *with* $O(h^3)$ *and* $O(h^4 \log h)$ *states, respectively. Hence* $\textsc{bgap} \in \mathsf{RO}/\mathsf{poly}$ *and* $\textsc{ugap} \in \mathsf{RO}/\mathsf{unary}$.

Proof To solve $\textsc{bgap}_h$, a ROFA $M_2 = \big([h] \cup [h]{\times}[h^2], \{\mathtt{0},\mathtt{1}\}, .\,, 0, h{-}1\big)$ implements the standard nondeterministic algorithm for graph accessibility. Visiting vertex $i \in [h]$ of the input graph is simulated by visiting $\vdash$ in state i. From there, M nondeterministically 'guesses' a 'correct' next vertex (i.e., one leading to $h{-}1$) among all $j \in [h]$ with $j \neq i$ and goes on to verify that arrow (i, j) is indeed present, by sweeping the input deterministically up to the bit in position $i{\cdot}h{+}j{+}1$ (using states of the form $(j, .)$) and checking that it is $\mathtt{1}$; if not, then it hangs (in this branch of the computation); otherwise, it continues the sweep in state j up to $\dashv$, where it either jumps back to $\vdash$ in j, if $j \neq h{-}1$, or falls off $\dashv$ into $h{-}1$, if $j = h{-}1$.

For $\textsc{ugap}_h$, a ROFA $M_1 = \big([h]{\cup}\bigcup_{i,j} C_{i,j}, \{\mathtt{0}\}, .\,, 0, h{-}1\big)$ uses the same algorithm as M_2. The implementation differs only in how M_1 verifies the presence of an arrow (i, j). For this, it uses a cycle of $p_{(i,j)} = p_{i\cdot h+j+1}$ states,

$$C_{i,j} := \{(i, j, k) \mid 0 \leq k < p_{(i,j)}\}\,,$$

where every state (i, j, k) transitions to state $(i, j, k{+}1 \bmod p_{(i,j)})$ on $\mathtt{0}$. Entering the cycle is possible only by a transition from state i to state $(i, j, 0)$ on $\vdash$, whereas exiting is possible only by a transition from $(i, j, 0)$ to state j on $\dashv$. As a result, computing from $\vdash$ and state i back to $\vdash$ and state j is possible iff $p_{(i,j)}$ divides the input length. The size of the automaton is

$$\begin{aligned}h + \sum_{i,j\in[h]} p_{(i,j)} &= h + \sum_{i,j\in[h]} p_{i\cdot h+j+1}\\ &\leq h + h^2{\cdot}p_{h^2} = h + h^2 O\big(h^2 \log(h^2)\big) = O(h^4 \log h)\,,\end{aligned}$$

where we use the fact that the k-th smallest prime number is $O(k \log k)$. [6] □

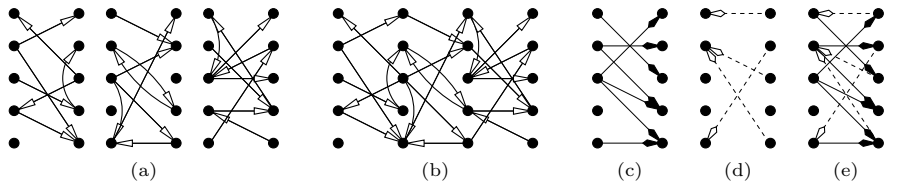

Fig. 2 (a) Three symbols of Γ_5. (b) The string of the symbols of (a) as a multi-column graph. (c) A symbol $(A,\text{L}) \in \Delta_5$. (d) A symbol $(B,\text{R}) \in \Delta_5'$. (e) The overlay of the symbols of (c) and (d).

2.3 Two-way liveness

The *two-way liveness* problem on height h, defined over the alphabet $\Gamma_h := \mathcal{P}(([h]\times\{\text{L},\text{R}\})^2)$ of all h-tall directed two-column graphs (Fig. 2a), is:

"Given a string x of graphs from Γ_h, check that x is live."

Here, $x \in \Gamma_h^*$ is *live* if the multi-column graph derived from x by identifying adjacent columns (Fig. 2b) contains 'live' paths, i.e., paths from the leftmost to the rightmost column; if not, then x is *dead*. Formally, this is the language

$$\text{TWL}_h := \{x \in \Gamma_h^* \mid x \text{ is live}\}\,.$$

We focus on two restrictions of this problem, in which x is promised to consist of $\leq h$ or of exactly 2 graphs. Formally, these are the promise problems

$$\text{SHORT TWL}_h := \big(\ \{x \in \Gamma_h^{\leq h} \mid x \text{ is live}\},\ \{x \in \Gamma_h^{\leq h} \mid x \text{ is dead}\}\ \big)$$
$$\text{COMPACT TWL}_h := \big(\ \{x \in \Gamma_h^{2} \mid x \text{ is live}\},\ \{x \in \Gamma_h^{2} \mid x \text{ is dead}\}\ \big)$$

and the families

$$\text{SHORT TWL} := (\text{SHORT TWL}_h)_{h\geq 1}$$
$$\text{COMPACT TWL} := (\text{COMPACT TWL}_h)_{h\geq 1}\,.$$

Lemma 3 TWL_h *is solved by a* 2NFA *with* $2h$ *states. Hence* SHORT TWL *and* COMPACT TWL *are both in* RO/poly.

Proof The $2h$-state 2NFA for TWL_h is straigthforward and well-known, and it clearly solves SHORT TWL_h and COMPACT TWL_h. So, SHORT TWL and COMPACT TWL are both in 2N/poly, and, by Corollary 1, also in RO/poly. □

2.4 Relational and functional match

The *relational match* problem on $[h]$ is defined over $\Delta_h := \mathbb{P}([h] \times [h])\times\{\text{L},\text{R}\}$, the alphabet of all pairs of binary relations on $[h]$ and side tags. A symbol (A,L) denotes an h-tall two-column graph with rightward arrows chosen by A (Fig. 2c); a symbol (B,R) denotes a similar graph with leftward arrows chosen

by B (Fig. 2d). If the overlay of these two graphs (Fig. 2e) contains a cycle, we say that the symbols *match*, or just that A, B *match*. We consider the problem

$$\begin{aligned}\textsc{rm}_h := \big(&\{\ (A,\textsc{l})(B,\textsc{r})\ \mid A,B \subseteq [h]\times[h]\ \ \&\ \ A,B \text{ match}\},\\ &\{\ (A,\textsc{l})(B,\textsc{r})\ \mid A,B \subseteq [h]\times[h]\ \ \&\ \ A,B \text{ do not match}\}\ \big)\end{aligned}$$

of checking that two given relations match. We also let $\textsc{fm}_h$ be the restriction of $\textsc{rm}_h$ to the alphabet $\Delta'_h := ([h]\rightharpoonup[h])\times\{\textsc{l},\textsc{r}\}$, where all relations are partial functions. We set

$$\textsc{rel match} := (\textsc{rm}_h)_{h\geq 1} \text{ and } \textsc{fun match} := (\textsc{fm}_h)_{h\geq 1}\,.$$

Lemma 4 $\textsc{fm}_h$ *is solved by a* 2DFA *with* h^3 *states. Hence* $\textsc{fun match} \in \mathsf{2D}$.

Proof To solve $\textsc{fm}_h$, a 2DFA $M = \big([h]^3, \Delta'_h, .\,, (0,0,0), (0,0,0)\big)$ searches exhaustively for a cycle: for every $i \in [h]$, it follows the unique path out of the i-th vertex of the left column in the overlay of the two input symbols, to check whether it returns to that vertex after $\leq 2h$ steps (clearly, if no cycle of length $\leq 2h$ exists, then no cycle of any length does). State (i,j,k) 'means' that the i-th search is currently on vertex j of the outer column of the current symbol and just before the $(k{+}1)$-th pair of successive steps. □

Finally, $\textsc{rel zero-match} = (\textsc{rzm}_h)_{h\geq 1}$ and $\textsc{fun zero-match} = (\textsc{fzm}_h)_{h\geq 1}$ are the variants where we only check whether the given relations or functions '*match through* 0', i.e., create a cycle *through vertex* 0 *of the left column*.

3 Transducers, reductions, and completeness

A *two-way deterministic finite transducer* (2DFT) consists of a finite control, an end-marked, read-only input tape accessed via a two-way head, and an infinite, write-only output tape accessed via a one-way head. Formally, it is a tuple $T = (S, \Sigma, \Gamma, \delta, q_0, q_\mathrm{f})$ of a set of states S, an input alphabet Σ, an output alphabet Γ, a start state $q_0 \in S$, a final state $q_\mathrm{f} \in S$, and a transition function

$$\delta : S \times (\Sigma \cup \{\vdash,\dashv\}) \to S \times \{\textsc{l},\textsc{r}\} \times \Gamma^*\,,$$

where $\vdash,\dashv \notin \Sigma$. An input $x \in \Sigma^*$ is presented on the input tape as $\vdash x \dashv$. The computation starts at q_0 with the input head on $\vdash$ and the output tape empty. At every step, T applies δ on the current state and input symbol to decide the next state, the next input head move, and the next string to append to the contents of the output tape; if this string is non-empty, the step is called *printing*. We assume δ never violates an end-marker, except if the input head is on $\dashv$ and the next state is q_f. For $y \in \Gamma^*$, we say T *outputs* y and write $T(x) = y$, if T eventually falls off $\dashv$ and then the output tape contains y. For $f : \Sigma^* \rightharpoonup \Gamma^*$ a partial function, we say T *computes* f if $T(x) = f(x)$ for all $x \in \Sigma^*$. If $\Gamma = \{\mathtt{0}\}$, then T can also 'compute' each unary string $f(x)$ by printing (not the string itself, but) an encoding of its length; if this is a prime

encoding $\#z_1\#z_2\#\cdots\#z_m$ (cf. p. 4) and every infix $\#z_i$ is printed by a single printing step (possibly together with other infixes), then T *prime-computes* f.

Now let $\mathfrak{L} = (L, \tilde{L})$ and $\mathfrak{L}' = (L', \tilde{L}')$ be two problems over alphabets Σ and Σ'. A (*mapping*) *reduction* of $\mathfrak{L}$ to $\mathfrak{L}'$ is any function $f : \Sigma^* \to (\Sigma')^*$ which is computable by a 2DFT and for every $x \in \Sigma^*$ satisfies

$$x \in L \Rightarrow f(x) \in L' \text{ and } x \in \tilde{L} \Rightarrow f(x) \in \tilde{L}' .$$

Furthermore:

- f is a *prime reduction* if $\Sigma' = \{\mathtt{0}\}$ and f is prime-computable by a 2DFT,
- f is a *homomorphic reduction* if $f(x) = g(\vdash)g(x_1)\cdots g(x_{|x|})g(\dashv)$ for some homomorphism $g : \Sigma \cup \{\vdash,\dashv\} \to (\Sigma')^*$ and all $x \in \Sigma^*$.

If such f exists, we say $\mathfrak{L}$ (*mapping-*)/*prime-*/*homomorphically reduces* to $\mathfrak{L}'$, and write $\mathfrak{L} \leq_{\mathrm{m}} \mathfrak{L}'$ / $\mathfrak{L} \preceq_{\mathrm{m}} \mathfrak{L}'$ / $\mathfrak{L} \leq_{\mathrm{h}} \mathfrak{L}'$. Easily, $\mathfrak{L} \leq_{\mathrm{h}} \mathfrak{L}'$ implies $\mathfrak{L} \leq_{\mathrm{m}} \mathfrak{L}'$.

Lemma 5 *If $\mathfrak{L}$ is solved by an s-state* 2OFA*, then*

$$\mathfrak{L} \leq_{\mathrm{m}} \mathrm{BGAP}_{2s+1} , \mathfrak{L} \preceq_{\mathrm{m}} \mathrm{UGAP}_{2s+1} , \textit{ and } \mathfrak{L} \leq_{\mathrm{h}} \mathrm{TWL}_{2s} .$$

The first two reductions are computable and prime-computable, respectively, by 2DFTs *with $O(s^4)$ states and $O(s^2)$ printing steps per input; the last reduction maps strings of length n to strings of length $n{+}2$.*

Proof Suppose $\mathfrak{L} = (L, \tilde{L})$ is solved by the s-state 2OFA $M = (S, \Sigma, ., q_0, q_{\mathrm{f}})$ and let $x \in \Sigma^*$. It is well-known that $\mathfrak{L} \leq_{\mathrm{h}} \mathrm{TWL}_{2s}$ via a reduction f with $|f(x)| = |x|{+}2$ (even if M is a general 2NFA; e.g., see [8, Lemma 3]). So, we focus on the first two claims.

A *segment* of M on x is a computation of M on x that starts and ends on an end-marker visiting no end-markers in between, or a single-step computation that starts on $\dashv$ and falls off into q_{f}. The *end-points* of a segment are its first and last configurations. The *summary* of M on x is a subgraph $G(x)$ of K_{2s+1} that encodes all segments, as follows. The vertices represent segment end-points: vertex 0 represents $(q_0, 0)$; vertices $1, 2, \ldots, 2s{-}1$ represent the remaining points of the form $(q, 0)$ and all points of the form $(q, |x|{+}1)$; and vertex $2s$ represents $(q_{\mathrm{f}}, |x|{+}2)$. Hence, each arrow of K_{2s+1} represents a possible segment. The summary $G(x)$ contains exactly those arrows which correspond to segments that M can perform on x. Easily, every accepting branch in the computation of M on x corresponds to a path in $G(x)$ from vertex 0 to vertex $2s$, and vice-versa.

Now let the functions $f_2(x){:=}\langle G(x)\rangle_2$ and $f_1(x){:=}\langle G(x)\rangle_1$ map every x to an instance of BGAP_{2s+1} and of UGAP_{2s+1}. If $x \in L$, then the computation of M on x contains an accepting branch, so $G(x)$ contains a path $0 \rightsquigarrow 2s$, thus $f_2(x)$ and $f_1(x)$ are positive instances. If $x \in \tilde{L}$, then there is no accepting branch, hence $G(x)$ contains no path $0 \rightsquigarrow 2s$, thus $f_2(x)$ and $f_1(x)$ are negative instances. So, f_2 and f_1 are the desired reductions, if they can be computed appropriately.

To compute $f_2(x)$ from $x \in \Sigma^*$, a 2DFT T_2 iterates over all arrows of K_{2s+1}; for each of them, it checks whether M can perform on x the corresponding segment, and outputs `1` or `0` accordingly. To check a segment, from end-point (p, i) to end-point (q, j), it iterates over all configurations (p', i') that are nondeterministically visited by M right after (p, i); for each of them, it checks whether M can compute from (p', i') to (q, j); the segment check succeeds if any of these checks does. Finally, to check the computation from (p', i') to (q, j), the transducer could simulate M from (p', i') and up to the first visit to an end-marker. This is indeed possible, since M would behave deterministically throughout this simulation. However, M could also loop, causing T_2 to loop as well, which is a problem.

To avoid this, T_2 simulates a halting variant M' of M, derived as follows.

1. We remove from M all transitions performed on $\vdash$ or $\dashv$.
2. We add a fresh start state q'_0, along with transitions which guarantee that, on its first transition leaving q'_0, the machine will be in state p' and cell i' (since cell i' is adjacent to either $\vdash$ or $\dashv$, this requires either a single step from q'_0 to p' on $\vdash$ or a full forward sweep in q'_0 followed by a single backward step on $\dashv$ into p').
3. We add a fresh final state q'_f, along with transitions which guarantee that, from state q reading the end-marker of cell j, the machine sweeps the tape in state q'_f until it falls off $\dashv$ (since cell j contains either $\vdash$ or $\dashv$, this is either a full forward sweep followed by a single step off $\dashv$, or just a single step off $\dashv$).

Now we have a 2DFA which first brings itself into configuration (p', i'), then simulates M until the first visit to an end-marker, and eventually accepts only if this simulation reaches (q, j). So, this is a $(2{+}s)$-state 2DFA that accepts x iff M can perform on x the segment from (p', i') to (q, j). By [4], this 2DFA has an equivalent *halting* 2DFA with $\leq 4{\cdot}(2{+}s)$ states. This is our M'.

To recap, T_2 iterates over all $O(s^2)$ arrows of K_{2s+1} and then over all $O(s)$ first steps of M in each corresponding segment, finally simulating a $O(s)$-state 2DFA in each iteration. Easily, T_2 needs no more than $O(s^4)$ states and $O(s^2)$ printing transitions, each used at most once. This proves our claim for f_2.

To prime-compute $f_1(x)$ from $x \in \Sigma^*$, a 2DFT T_1 must print a prime encoding $\mathtt{\#}z_1 \cdots \mathtt{\#}z_m$ of the length of $\langle G(x)\rangle_1$ (also making sure no infix $\mathtt{\#}z_i$ is split between printing steps). By (2), this length is the product of the primes p_k for which the k-th bit of $\langle G(x)\rangle_2$ is `1`. So, m must equal the number of `1`s in $T_2(x)$ and each z_i must encode one of the trivial prime powers of the form p_k^1 corresponding to those `1`s. Hence, T_1 simulates T_2 and, every time T_2 would print a `1` as its k-th ouput bit, T_1 performs a printing step with output $\mathtt{\#}z$, where z is the encoding of p_k^1. Easily, the state diagram of T_1 differs from that of T_2 only in the output strings on the printing transitions. So, the size and print of T_1 are also $O(s^4)$ and $O(s^2)$. □

For two families $\mathcal{T}=(T_h)_{h\geq 1}$ of 2DFTs and $\mathcal{F}=(f_h)_{h\geq 1}$ of string functions, we say $\mathcal{T}$ (*prime-*) *computes* $\mathcal{F}$ if every T_h (prime-) computes f_h. Furthermore:

- the members of $\mathcal{T}$ are *laconic* if every T_h performs $\leq p(h)$ printing steps on each input, for some polynomial p, and
- the members of $\mathcal{F}$ are *tight* if $|f_h(x)| \leq p(h)\cdot|x|$ for some polynomial p, all h, and all x.

For two problem families $\mathcal{L}=(\mathfrak{L}_h)_{h\geq 1}$ and $\mathcal{L}'=(\mathfrak{L}'_h)_{h\geq 1}$, we say $\mathcal{L}$ *reduces to* $\mathcal{L}'$ *in polynomial size* ($\mathcal{L} \leq_{\mathsf{2D}} \mathcal{L}'$) if every $\mathfrak{L}_h$ reduces to $\mathfrak{L}'_{p(h)}$ for some polynomial p, and the family $\mathcal{F}$ of underlying reductions is computed by a family $\mathcal{T}$ of small 2DFTs. Furthemore:

- $\mathcal{L}$ *prime-reduces to* $\mathcal{L}'$ *in polynomial size* ($\mathcal{L} \preceq_{\mathsf{2D}} \mathcal{L}'$) if the problems in $\mathcal{L}'$ are unary and $\mathcal{T}$ prime-computes $\mathcal{F}$,
- $\mathcal{L}$ (*prime-*) *reduces to* $\mathcal{L}'$ *in polynomial size/print* ($\mathcal{L} \leq^{\text{lac}}_{\mathsf{2D}} \mathcal{L}'$ or $\mathcal{L} \preceq^{\text{lac}}_{\mathsf{2D}} \mathcal{L}'$) if the 2DFTs in $\mathcal{T}$ are laconic,
- $\mathcal{L}$ *homomorphically reduces to* $\mathcal{L}'$ ($\mathcal{L} \leq_{\text{h}} \mathcal{L}'$) if every $\mathfrak{L}_h$ homomorphically reduces to $\mathfrak{L}'_{p(h)}$,
- $\mathcal{L}$ *reduces to* $\mathcal{L}'$ *under tight homomorphisms* ($\mathcal{L} \leq^{\text{t}}_{\text{h}} \mathcal{L}'$) if $\mathcal{L} \leq_{\text{h}} \mathcal{L}'$ and the underlying homomorphisms are tight.

As usual, if C is a class of problem families and $\leq$ a type of reductions, then a family $\mathcal{L}$ is C-*complete under* $\leq$ if $\mathcal{L} \in \mathsf{C}$ and every $\mathcal{L}' \in \mathsf{C}$ satisfies $\mathcal{L}' \leq \mathcal{L}$.

Corollary 2 *The following statements are true:*

1. BGAP *is* $\mathsf{2N/poly}$*-complete and* $\mathsf{2O}$*-complete, under polynomial-size/print reductions* ($\leq^{\text{lac}}_{\mathsf{2D}}$).
2. UGAP *is* $\mathsf{2N/unary}$*-complete and* $\mathsf{2O}$*-complete, under polynomial-size/print prime reductions* ($\preceq^{\text{lac}}_{\mathsf{2D}}$).
3. SHORT TWL *is* $\mathsf{2N/poly}$*-complete under tight homomorphic reductions* ($\leq^{\text{t}}_{\text{h}}$).

Proof The required memberships, of BGAP in $\mathsf{2N/poly}$ and $\mathsf{2O}$, of UGAP in $\mathsf{2N/unary}$ and $\mathsf{2O}$, and of SHORT TWL in $\mathsf{2N/poly}$ follow from Lemmas 2 and 3.

For the hardness of BGAP and UGAP, pick any $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1} \in \mathsf{2N/poly} \cup \mathsf{2N/unary} \cup \mathsf{2O}$. By Corollary 1, we know $\mathcal{L} \in \mathsf{2O}$. So, every $\mathfrak{L}_h$ is solved by an s-state 2OFA, where $s = \text{poly}(h)$. By Lemma 5, this implies $\mathfrak{L}_h \leq_{\text{m}} \text{BGAP}_{p(h)}$, where $p(h) := 2s + 1 = \text{poly}(h)$ and the underlying reduction f_h is computed by a 2DFT T_h with $O(s^4) = \text{poly}(h)$ states and $O(s^2) = \text{poly}(h)$ printing steps on each input. So, $\mathcal{F}:=(f_h)_{h\geq 1}$ is computed by the small and laconic 2DFTs of $\mathcal{T}:=(T_h)_{h\geq 1}$, hence $\mathcal{L} \leq^{\text{lac}}_{\mathsf{2D}}$ BGAP. Lemma 5 also implies that $\mathfrak{L}_h \preceq_{\text{m}} \text{UGAP}_{p(h)}$, where the underlying reduction f'_h is prime-computed by a 2DFT T'_h with $O(s^4) = \text{poly}(h)$ states and $O(s^2) = \text{poly}(h)$ printing steps; so, $\mathcal{F}':=(f'_h)_{h\geq 1}$ is prime-computed by the small and laconic 2DFTs of $\mathcal{T}':=(T'_h)_{h\geq 1}$. So, $\mathcal{L} \preceq^{\text{lac}}_{\mathsf{2D}}$ UGAP.

For the hardness of SHORT TWL, suppose $\mathcal{L} \in \mathsf{2N/poly}$. Then every $\mathcal{L}_h$ is solved by an s-state 2NFA and its instances have lengths $\leq l$, where $s, l = \text{poly}(h)$. By Lemma 5, we know $\mathfrak{L}_h \leq_{\text{h}} \text{TWL}_{2s}$ via a homomorphism f_h such

that $|f_h(x)| = |x| + 2 \leq l + 2$, for all instances x. Letting $m{:=}\max(2s, l+2)$, we have $\mathfrak{L}_h \leq_{\rm h} \text{TWL}_m$, via the modification f'_h of f_h which returns the same strings of graphs but with $m - 2s$ additional rows, of isolated vertices. Since $|f'_h(x)| = |f_h(x)| \leq l + 2 \leq m$, every instance of TWL_m returned by f'_h is also an instance of SHORT TWL_m. Therefore, f'_h proves that $\mathfrak{L}_h \leq_{\rm h}$ SHORT TWL_m. Since $m = \text{poly}(h)$ and the homomorphisms in $\mathcal{F}'{:=}(f'_h)_{h\geq 1}$ are tight (clearly), we conclude that $\mathcal{L} \leq^{\rm t}_{\rm h}$ SHORT TWL. □

4 Closures

We now prove that 2D is closed under all of the above reductions. As usual, a class C is *closed* under a type $\leq$ of reductions if $\mathcal{L}_1 \leq \mathcal{L}_2$ & $\mathcal{L}_2 \in \mathsf{C} \Rightarrow \mathcal{L}_1 \in \mathsf{C}$.

Lemma 6 *Suppose $\mathfrak{L}_2$ is solved by an s-state* 2DFA*. Then the following hold.*

1. *If $\mathfrak{L}_1 \leq_{\rm m} \mathfrak{L}_2$ via a reduction which is computable by an r-state* 2DFT *performing $\leq t$ printing steps on every input, then $\mathfrak{L}_1$ is solved by a* 2DFA *with $O(rst^2)$ states.*
2. *If $\mathfrak{L}_1 \preceq_{\rm m} \mathfrak{L}_2$ via a reduction which is prime-computable by an r-state* 2DFT*, then $\mathfrak{L}_1$ is solved by a* 2DFA *with $O(rs^2)$ states.*
3. *If $\mathfrak{L}_1 \leq_{\rm h} \mathfrak{L}_2$, then $\mathfrak{L}_1$ is solved by a* 2DFA *with $2s$ states.*

Proof Part 3 is well-known (e.g., see [8, Lemma 2]), so we focus on the first two claims, starting with 1. Suppose $f : \Sigma_1^* \to \Sigma_2^*$ reduces $\mathfrak{L}_1$ to $\mathfrak{L}_2$. Let $T = (S, \Sigma_1, \Sigma_2, .\,, .\,, .\,)$ be a 2DFT that computes f, with $|S| = r$ and $\leq t$ printing steps on every input. Let $M_2 = (S_2, \Sigma_2, .\,, .\,, .\,)$ be a 2DFA that solves $\mathfrak{L}_2$, with $|S_2| = s$. We build a 2DFA $M_1 = (S_1, \Sigma_1, .\,, .\,, .\,)$ for $\mathfrak{L}_1$.

On input $x \in \Sigma_1^*$, M_1 simulates T on x to compute $f(x)$, then simulates M_2 on $f(x)$ and copies its decision. (By the choice of f, this is clearly a correct algorithm for $\mathfrak{L}_1$.) Of course, M_1 cannot store $f(x)$ in between the two simulations. So, it performs them simultaneously: it simulates T on x and, every time T would print a string z (an infix of $f(x)$), M_1 resumes the simulation of M_2 from where it was last interrupted and for as long as it stays within z.

The only problem (a classic, from space complexity) is that T prints $f(x)$ by a one-way head but M_2 reads $f(x)$ by a two-way head. So, whenever the simulation of M_2 falls off the *left* boundary of an infix z, the simulation of T should not continue unaffected to return the next infix after z, but should return again the infix *before* z. The solution (also a classic) is to restart the simulation of T, continue until the desired infix is ready to be output, and then resume the simulation of M_2. This is possible exactly because of the bound t, which implies that $f(x) = z_1 z_2 \cdots z_m$ where $m \leq t$ and z_i is the infix output by T in its i-th printing step. Thus M_1 keeps track of the index i of the infix currently read by the simulation of M_2, and uses it to request z_{i-1} from the next simulation of T.

Notice that M_1 does not need to explicitly represent the infix currently scanned by the simulation of M_2. In fact, the infix z produced by T in a printing step depends only on the current state $p \in S$ and input symbol

$a \in \Sigma_1 \cup \{\vdash,\dashv\}$. Furthermore, for $(q,d) \in S_2 \times \{\text{L},\text{R}\}$, let $(q',d') \in S_2 \times \{\text{L},\text{R}\}$ be such that M_2 from state q on the d-most symbol of z, after a sequence of steps, will finally leave z with a transition from the d'-most symbol, entering state q'. Hence (q',d'), if exists, is uniquely determined from the pairs (p,a) and (q,d). This means that a sequence of consecutive transitions of M_2, which starts by entering some infix z produced by a single step of T and ends by leaving the same z, can be simulated by a single transition of M_1, which does not explicitly represent z.

Easily, M_1 implements the above algorithm with $S_1 := S_2 \times \{\text{L},\text{R}\} \times [t] \times S \times [t]$, where state (q,d,i,p,j) 'means' that the simulation of M_2 is ready to resume from state q on the d-most symbol of z_{i+1}, and the simulation of T (to output z_{i+1}) is in state p and past the printing steps for $z_1,\ldots,z_j$. As claimed, $|S_1| = O(rst^2)$.

Now suppose $\Sigma_2 = \{\mathtt{0}\}$ and T prime-computes f. Then M_1 implements a modification of the above algorithm. Now every string returned by the simulation of T is an infix not of $f(x)$ but of a prime encoding $\mathtt{\#}z_1\mathtt{\#}z_2\mathtt{\#}\cdots\mathtt{\#}z_m$ of $n := |f(x)|$. So, we need to change the way these infixes are treated by the simulation of M_2.

By the analyses of [9,3], it follows that there exist a length bound $l = O(s)$ and a $O(s)$-state RDFA $\tilde{M}_2 = (\tilde{S}_2, \{\mathtt{0}\}, \tilde{\delta}, \tilde{q}_0, \tilde{q}_\text{f})$ such that $\tilde{M}_2$ agrees with M_2 on all lengths $\geq l$, and is in the following 'normal form'. The states $\tilde{S}_2 \setminus \{\tilde{q}_0,\tilde{q}_\text{f}\}$ can be partitioned into a number of cycles $C_1, C_2, \ldots, C_k$. Every rotation on an input y starts on $\vdash$ with a transition into a state $\tilde{q}$ of some C_j, and finishes on $\dashv$ in the state $\tilde{p}$ of the same C_j that is uniquely determined by the entry state $\tilde{q}$, the cycle length $|C_j|$, and the input length $|y|$. From there, the next transition is either off $\dashv$ into $\tilde{q}_\text{f}$ to accept, or back to $\vdash$ and again in $\tilde{p}$ to start a new rotation. Our modified M_1 will use this $\tilde{M}_2$ for checking whether M_2 accepts $f(x)$.

In a first stage, M_1 checks whether $n = |f(x)|$ is one of the lengths $< l$, where $\tilde{M}_2$ and M_2 may disagree; if so, then it halts, accepting iff M_2 accepts $\mathtt{0}^n$; otherwise, it continues to the second stage below. To check whether $n < l$, it iterates over all $\hat{n} \in [l]$, checking for each of them whether $n = \hat{n}$. To check whether $n = \hat{n}$, it checks whether the prime factorizations of the two numbers contain the same prime powers. This needs $1+\hat{m}$ simulations of T on x, for $\hat{m} \leq \log \hat{n}$ the number of prime powers in the factorization of $\hat{n}$: during the 0-th simulation, M_1 checks that every infix $\mathtt{\#}z_i$ of every string output by T encodes some prime power of $\hat{n}$; during the j-th simulation ($1 \leq j \leq \hat{m}$), M_1 checks that the j-th prime power of $\hat{n}$ (under some fixed ordering of prime powers) is encoded by some infix $\mathtt{\#}z_i$ of some string output by T.[5] Overall, this first stage can be implemented on the state set $[l] \times [1+\log l] \times S$, where state $(\hat{n},j,q)$ 'means' that the j-th simulation of T for checking $n = \hat{n}$ is currently in state q.

[5] Notice that even in this case, the string produced by T in a printing step (which contains one or more infixes $\mathtt{\#}z_i$) does not need to be explicitly represented: it only depends on the simulated printing step of T.

In the second stage, $n \geq l$ is guaranteed, so M_1 can simulate $\tilde{M}_2$ instead of M_2. This is done one rotation at a time. Starting the simulation of a single rotation, M_1 knows the entry state $\tilde{q}$ and cycle C_j to be used; the goal is to compute the state $\tilde{p}$ of C_j when the rotation reaches $\dashv$. Clearly, this reduces to computing the remainder $n \bmod l_j$, where $l_j{:=}|C_j|$ is the cycle length. If n_i is the prime power encoded by the infix $\#z_i$ of $T(x)$, then $n \bmod l_j$ equals

$$\Big(\prod_{i=1}^{m} n_i\Big) \bmod l_j = \big(\big(\cdots\big((n_1 \bmod l_j)\cdot n_2\big) \bmod l_j \cdots\big)\cdot n_m\big) \bmod l_j\,. \quad (3)$$

So, to compute the value ρ of this remainder, M_1 simulates T on x, building ρ as in (3): $\rho \leftarrow 1$ at start, and $\rho \leftarrow (\rho \cdot n_i) \bmod l_j$ for every infix $\#z_i$ inside a string printed by T.[6] When the simulation of T halts, M_1 finds $\tilde{p}$ in C_j at distance ρ from $\tilde{q}$. From there, if $\tilde{\delta}(\tilde{p},\dashv) = \tilde{q}_{\mathrm{f}}$ then $\tilde{M}_2$ accepts, and so does M_1; otherwise, $\tilde{M}_2$ starts a new rotation from state $\tilde{\delta}(\tilde{\delta}(\tilde{p},\dashv),\vdash)$, and M_1 goes on to simulate it, too.

Overall, the second stage can be implemented on the state set $\tilde{S}_2 \times S \times [\,|\tilde{S}_2|\,]$, where state $(\tilde{q}, q, \rho)$ 'means' that the simulation of T for simulating a rotation of $\tilde{M}_2$ from state $\tilde{q}$ is currently in state q and the remainder is currently ρ.

Summing up, $S_1{:=}([l] \times [1+\log l] \times S) \cup (\tilde{S}_2 \times S \times [\,|\tilde{S}_2|\,])$. Hence $|S_1| = O(rs^2)$. □

Corollary 3 2D *is closed under polynomial-size/print reductions* ($\leq^{\mathrm{lac}}_{\mathsf{2D}}$)*, under polynomial-size prime reductions* ($\preceq_{\mathsf{2D}}$)*, and under homomorphic reductions* ($\leq_{\mathrm{h}}$)*.*

Proof Pick any two problem families $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1}$ and $\mathcal{L}' = (\mathfrak{L}'_h)_{h\geq 1}$, and suppose $\mathcal{L}' \in$ 2D. Then every $\mathfrak{L}'_h$ is solved by a $p(h)$-state 2DFA, for some polynomial p.

If $\mathcal{L} \leq^{\mathrm{lac}}_{\mathsf{2D}} \mathcal{L}'$ then every $\mathfrak{L}_h$ reduces to an $\mathfrak{L}'_{h'}$ via a reduction computable by an r-state 2DFT performing $\leq t$ printing steps on every input, where $h', r, t = \mathrm{poly}(h)$. By Lemma 6.1, it follows that $\mathfrak{L}_h$ is solved by a 2DFA with $O\big(r{\cdot}p(h'){\cdot}t^2\big) = \mathrm{poly}(h)$ states. So, $\mathcal{L} \in$ 2D.

If $\mathcal{L} \preceq_{\mathsf{2D}} \mathcal{L}'$ then every $\mathfrak{L}_h$ prime-reduces to an $\mathfrak{L}'_{h'}$ via a reduction prime-computable by an r-state 2DFT, where $h', r = \mathrm{poly}(h)$. By Lemma 6.2, it follows that $\mathfrak{L}_h$ is solved by a 2DFA with $O\big(r{\cdot}p(h')^2\big) = \mathrm{poly}(h)$ states. So, $\mathcal{L} \in$ 2D.

If $\mathcal{L} \leq_{\mathrm{h}} \mathcal{L}'$ then every $\mathfrak{L}_h$ homomorphically reduces to an $\mathfrak{L}'_{h'}$, where $h' = \mathrm{poly}(h)$. By Lemma 6.3, it follows that $\mathfrak{L}_h$ is solved by a 2DFA with $2{\cdot}p(h') = \mathrm{poly}(h)$ states. So, $\mathcal{L} \in$ 2D. □

[6] When a printing step of T produces an infix $\#z_i\#z_{i+1}\cdots\#z_k$, M_1 can compute in a single step $\big(\cdots\big((\rho{\cdot}n_i \bmod l_j){\cdot}n_{i+1}\big) \bmod l_j \cdots\big){\cdot}n_k\big) \bmod l_j$. Since the infix depends only on the printing step of T, this can be embedded in the transition function of M_1, without explicitly representing the infix $\#z_i\#z_{i+1}\cdots\#z_k$.

5 Characterizations and conjectures

Our main theorem is now a direct consequence of [8] and Corollaries 2 and 3.

Theorem 1 *The following statements are equivalent to* $\mathsf{L/poly} \supseteq \mathsf{NL}$*:*

1. $\mathsf{2D} \supseteq \mathsf{2N/poly}$ **3.** $\mathsf{2D} \supseteq \mathsf{2N/unary}$ **5.** $\mathsf{2D} \supseteq \mathsf{2O}$

2. $\mathsf{2D} \ni$ BGAP **4.** $\mathsf{2D} \ni$ UGAP **6.** $\mathsf{2D} \ni$ SHORT TWL

Proof We have $\mathsf{L/poly} \supseteq \mathsf{NL}$ iff (1), by [8]; (1)⇔(2)⇔(5), by Corollary 2.1 and the closure of 2D under polynomial-size/print reductions; (3)⇔(4)⇔(5) by Corollary 2.2 and the closure of 2D under polynomial-size prime reductions; and (1)⇔(6) by Corollary 2.3 and the closure of 2D under homomorphic reductions. □

The statements of Theorem 1 are believed to be false. In particular (statement 6), it is conjectured that no poly(h)-state 2DFA can check liveness on inputs of height h and length $\leq h$. It is thus natural to study shorter inputs. In fact, even inputs of length 2 are interesting:

How large need a 2DFA *be to solve* COMPACT TWL$_h$*?*

Although it can be shown (by Savitch's algorithm [11]) that $2^{O(\log^2 h)}$ states are enough, it is not known whether this can be reduced to poly(h). We conjecture that it cannot. In other words, we conjecture that $\mathsf{L/poly} \not\supseteq \mathsf{NL}$ because already:

Conjecture A $\mathsf{2D} \not\ni$ COMPACT TWL.

We find this conjecture quite appealing, because of its simple and symmetric setup: just one 2DFA working on just two symbols. Still, this is an *algorithmic* statement (it claims the inexistence of an algorithm), engaging our algorithmic intuitions. These are surely welcome when we need to discover an algorithm, but may be unfavorable when we need to prove that no algorithm exists. It could thus be advantageous to complement this statement with an equivalent one which is purely *combinatorial*. Indeed, such a statement exists: it says that we cannot homomorphically reduce relational to functional match (cf. p. 9).

Conjecture B REL MATCH $\not\leq_{\mathrm{h}}$ FUN MATCH.

To get a feel of this conjecture, it is useful to note first that RM$_h \leq_{\mathrm{h}}$ FM$_{2^{h^2}}$,[7] and then that this does not imply REL MATCH $\leq_{\mathrm{h}}$ FUN MATCH, because of the super-polynomial blow-up from h to 2^h. So, Conjecture B claims that this blow-up cannot be made poly(h) or, more intuitively, that *no systematic way*

[7] Fix any bijection π from the binary relations on $[h]$ to the numbers in $[2^{h^2}]$. Then the homomorphic image $f(A,\mathrm{L})$ of a left symbol (A,L) is the partial function that only maps the number $\pi(A)$ to itself. The image $f(B,\mathrm{R})$ of a right symbol (B,R) is the partial function in which a number i can only map to itself, and this holds iff $i = \pi(A)$ for some A that matches with B. Clearly then, A, B match iff $f(A,\mathrm{L}), f(B,\mathrm{R})$ match (by the one two-step cycle going from $\pi(A)$ on the left column to $\pi(A)$ on the right column and back).

of replacing h-tall relations with poly(h)*-tall functions respects the existence of cycles.*

To prove the equivalence of the two conjectures, we first show that checking for cycles is $\leq_{\mathrm{h}}$-equivalent to checking for cycles through the top left vertex.

Lemma 7 *The following reductions hold:*
1. REL ZERO-MATCH $\leq_{\mathrm{h}}$ REL MATCH *and* REL MATCH $\leq_{\mathrm{h}}$ REL ZERO-MATCH.
2. FUN ZERO-MATCH $\leq_{\mathrm{h}}$ FUN MATCH *and* FUN MATCH $\leq_{\mathrm{h}}$ FUN ZERO-MATCH.

Proof For part 1, we prove that $\mathrm{RZM}_h \leq_{\mathrm{h}} \mathrm{RM}_{2h^2}$ and that $\mathrm{RM}_h \leq_{\mathrm{h}} \mathrm{RZM}_{1+h^3}$. Both homomorphisms map $\vdash$ and $\dashv$ to ϵ, every (A,L) to a $(\tilde{A},\mathrm{L})$, and every (B,R) to a $(\tilde{B},\mathrm{R})$.

For the first reduction, we divide $[2h^2]$ into $2h$ levels of h items each, and refer to items by level index $l \in [2h]$ and index-within-level $i \in [h]$; i.e., we view $[2h^2]$ as $[2h]\times[h]$. Then:

- Every $(A,\mathrm{L}) \in \Delta_h$ maps to $(\tilde{A},\mathrm{L}) \in \Delta_{2h^2}$, where $\tilde{A}$ consists of h 'copies' of A: on each level of *even* index l, every arrow $i{\to}j \in A$ induces the arrow $(l,i){\to}(l+1,j)$, with the corresponding end-points on levels l and $l+1$.
- Symmetrically, every (B,R) maps to $(\tilde{B},\mathrm{R})$, where $\tilde{B}$ copies B: on each level of *odd* index l, every arrow $j{\leftarrow}i \in B$ with $j \neq 0$ induces the arrow $(l{+}1,j){\leftarrow}(l,i)$; in addition, every arrow $0{\leftarrow}i$ induces the arrow $(0,0){\leftarrow}(l,i)$.

Clearly, every arrow in the overlay of $(\tilde{A},\mathrm{L})$ and $(\tilde{B},\mathrm{R})$ either increases the level index or points to the left-column vertex $(0,0)$. (For an example, see Fig. 3c-d.) Hence, this vertex is visited by all cycles (if any) in the overlay, so $\tilde{A},\tilde{B}$ *match iff they match through* 0. Moreover, the overlay of (A,L) and (B,R) contains a k-long cycle through the left-column vertex 0 iff the overlay of $(\tilde{A},\mathrm{L})$ and $(\tilde{B},\mathrm{R})$ contains a k-long cycle through the left-column vertex $(0,0)$, where the t-th step moves from level $t{-}1$ to level t, and the last step moves from level $k{-}1$ to level 0 (here $1 \leq t < k \leq 2h$). So, A,B match through 0 iff $\tilde{A},\tilde{B}$ do, and thus iff $\tilde{A},\tilde{B}$ match.

For the second reduction, we divide $[1+h^3]$ into $1+h^2$ levels: one single-item level, plus h^2 levels of h items each. The item of the first level is called 0, and the items of the other levels are called by level index $(i,j) \in [h]\times[h]$ and index-within-level $u \in [h]$; i.e., we view $[1{+}h^3]$ as $\{0\}\cup([h]\times[h])\times[h]$. Then:

- Every $(A,\mathrm{L}) \in \Delta_h$ maps to $(\tilde{A},\mathrm{L}) \in \Delta_{1+h^3}$, where $\tilde{A}$ contains h^2 copies of A: on each level (i,j), every arrow $u{\to}v \in A$ induces the arrow $(i,j,u){\to}(i,j,v)$ of the corresponding end-points; in addition, the arrows $0{\to}(i,j,j)$ and $(i,j,i){\to}0$ are added whenever $i{\to}j \in A$.
- Symmetrically, every (B,R) maps to $(\tilde{B},\mathrm{R})$, where $\tilde{B}$ contains the arrow $0{\leftarrow}0$, plus h^2 copies of B: on each level (i,j), every arrow $v{\leftarrow}u \in B$ induces the arrow $(i,j,v){\leftarrow}(i,j,u)$.

Clearly, every cycle in the overlay of (A,L) and (B,R) copies itself on every h-item level of the overlay of $(\tilde{A},\mathrm{L})$ and $(\tilde{B},\mathrm{R})$; in addition, every forward arrow $i{\to}j \in A$ of the cycle induces the cycle

$$0 \xrightarrow{\tilde{A}} (i,j,j) \xrightarrow{\tilde{B}} \cdots \xrightarrow{\tilde{B}} (i,j,i) \xrightarrow{\tilde{A}} 0 \xrightarrow{\tilde{B}} 0\,, \tag{4}$$

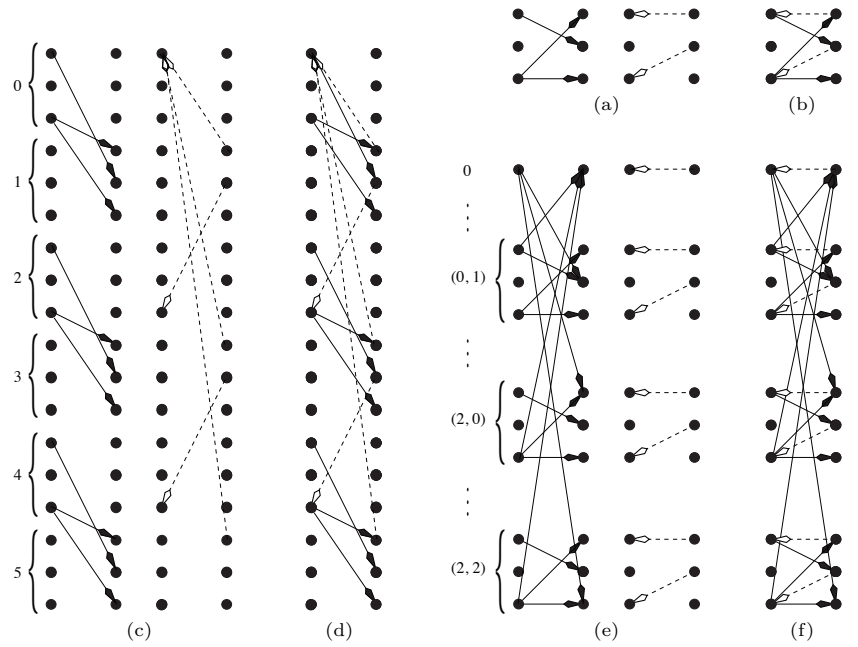

Fig. 3 Proof of Lemma 7.1. (a) Two symbols $(A,\text{L}), (B,\text{R}) \in \Delta_3$. (b) The overlay of the symbols of (a). (c) The symbols $(\tilde{A},\text{L}), (\tilde{B},\text{R}) \in \Delta_{18}$ obtained by the reduction $\text{RZM}_h \leq_\text{h} \text{RM}_{2h^2}$. (d) The overlay of the symbols of (c). (e) The symbols $(\tilde{A},\text{L}), (\tilde{B},\text{R}) \in \Delta_{28}$ obtained by the reduction $\text{RM}_h \leq_\text{h} \text{RZM}_{1+h^3}$. Only levels 0, $(0,1)$, $(2,0)$ and $(2,2)$ are represented. The other levels are not connected to level 0. (f) The overlay of symbols of (e). Notice that the cycle in the overlay of (A,L) and (B,R) (see (b)) induces several cycles in the overlay of $(\tilde{A},\text{L})$ and $(\tilde{B},\text{R})$, only two of which pass through the top-left vertex; e.g., $0 \longrightarrow^{\tilde{A}} (2,0,0) \longrightarrow^{\tilde{B}} (2,0,0) \longrightarrow^{\tilde{A}} (2,0,1) \longrightarrow^{\tilde{B}} (2,0,2) \longrightarrow^{\tilde{A}} 0 \longrightarrow^{\tilde{B}} 0$.

where '$\cdots$' stands for the copies of the remaining arrows of the cycle on level (i,j). (For an example, see Fig. 3e-f.) Conversely, if the overlay of $(\tilde{A},\text{L})$ and $(\tilde{B},\text{R})$ contains a cycle through vertex 0 of the left column, then this is necessarily of the form (4) for some arrow $i{\to}j \in A$, and for '$\cdots$' a path within level (i,j) which copies a path from right-column vertex j to left-column vertex i in the overlay of (A,L) and (B,R). Overall, A, B match iff $\tilde{A}, \tilde{B}$ match through 0.

For part 2, we prove that $\text{FZM}_h \leq_\text{h} \text{FM}_{2h^2}$ and $\text{FM}_h \leq_\text{h} \text{FZM}_{h^3}$. The first reduction follows simply by the observation that the first homomorphism of part 1 maps functions to functions: if A and B are functions, then $\tilde{A}$ and $\tilde{B}$ are functions as well. The second reduction follows from Lemma 4 and the argument in the proof of Lemma 9 below: the homomorphism can be extracted directly from the h^3-state 2DFA solving FM_h. (Note that the second reduction of part 2 is not really needed in our arguments; it is included only for completeness.) □

Using Lemma 7, we now prove that COMPACT TWL and REL MATCH reduce to each other in such a way that small 2DFAs can solve either both or neither (Lemma 8), and that REL MATCH is solvable by small 2DFAs iff it $\leq_{\mathrm{h}}$-reduces to FUN MATCH (Lemma 9).

Lemma 8 *The following reductions hold:*
COMPACT TWL $\leq_{\mathrm{2D}}^{\mathrm{lac}}$ REL MATCH *and* REL MATCH $\leq_{\mathrm{h}}$ COMPACT TWL.

Proof We first show COMPACT $\mathrm{TWL}_h \leq_{\mathrm{m}} \mathrm{RZM}_{1+h}$ by a 2DFT which simply sweeps its input $\vdash ab \dashv$ once, and replaces a with $(A,\mathrm{L}) \in \Delta_{1+h}$ and b with $(B,\mathrm{R}) \in \Delta_{1+h}$, such that ab is live iff A, B match through 0. Then, we combine this 2DFT with the homomorphism from the first half of Lemma 7.1, to get a new 2DFT which instead prints $(\tilde{A},\mathrm{L}), (\tilde{B},\mathrm{R}) \in \Delta_{2(1+h)^2}$ such that A, B match through 0 iff $\tilde{A}, \tilde{B}$ match. So, COMPACT $\mathrm{TWL}_h \leq_{\mathrm{m}} \mathrm{RM}_{2(1+h)^2}$ by a 2-state 2DFT with 2 printing steps per input, and we are done.

We derive A and B from a and b respectively, in two steps (see Fig. 4 for an example):

- First, we convert a, b into $a', b' \in \Gamma_{1+h}$ such that ab is live iff $a'b'$ contains a cycle which visits vertex 0 of the middle column and alternates between a' and b'.
- Then, we convert a' and b' to A and B.

To get a', we start with a and:
(i) drop all arrows that end on the left column, as they do not affect liveness;
(ii) introduce a fresh vertex 0 in both columns, to get a symbol of Γ_{1+h};
(iii) transfer to this vertex of the right column the origins of all arrows that begin on the left column, so that now all arrow end-points lie on the right column, and a' is essentially a binary relation on $[1+h]$;
(iv) add all arrows that are necessary to make this relation transitive.

Symmetrically, to get b', we start with b and:
(i) drop all arrows that begin on the right column;
(ii) introduce a fresh vertex 0 in both columns;
(iii) transfer to this vertex of the left column the destinations of all arrows that end on the right column, so that now all arrow end-points lie on the left column, and b' is essentially a binary relation on $[1+h]$;
(iv) add all arrows that are necessary to make this relation transitive.

Now, it is straightforward to see that every live path in ab induces a cycle in $a'b'$ through vertex 0 of the middle column, and vice-versa; moreover, because of transitivity, this cycle can always be chosen to alternate between a' and b'. Now, the relations A and B are exactly the binary relations on $[1+h]$ that describe the arrows of a' and b': $(i,j) \in A$ iff arrow $i \to j$ is present on the right column of a', and similarly for B and the left column of b'. Easily, A, B match through 0 iff a', b' contain and alternating cycle through vertex 0 of the middle column, and thus iff ab is live.

We now show $\mathrm{RZM}_h \leq_{\mathrm{h}}$ COMPACT TWL_{2h}. Combined with the second half of Lemma 7.1, this implies $\mathrm{RM}_h \leq_{\mathrm{h}}$ COMPACT $\mathrm{TWL}_{2(1+h^3)}$, and we are done. The homomorphism maps $\vdash$ and $\dashv$ to ϵ. For the other symbols, we divide $[2h]$

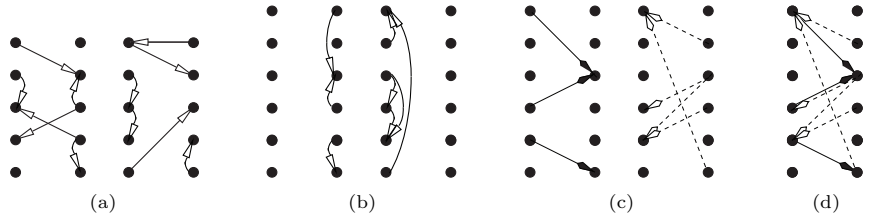

Fig. 4 COMPACT TWL $\leq^{\text{lac}}_{\text{2D}}$ REL MATCH (Lemma 8). (a) Two symbols $a, b \in \Gamma_5$. (b) The symbols $a', b' \in \Gamma_6$ obtained from a and b. (c) The symbols $(A,\text{L}), (B,\text{R}) \in \Delta_6$ obtained from a' and b'. (d) The overlay of the symbols of (c).

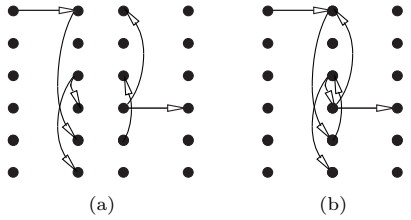

Fig. 5 REL MATCH $\leq_{\text{h}}$ COMPACT TWL (Lemma 8). (a) The symbols $a, b \in \Gamma_6$ obtained from the symbols $(A,\text{L}), (B,\text{R}) \in \Delta_3$ of Fig. 3a. (b) The string ab as a multi-column graph.

into two levels of h items each, calling each item by level index $l \in [2]$ and index-within-level $i \in [h]$; i.e., we view $[2h]$ as $[2]\times[h]$. Then, each $(A,\text{L}) \in \Delta_h$ maps to an $a \in \Gamma_{2h}$ whose right column encodes A, using level 0 for origins and level 1 for destinations: each arrow $i\to j \in A$ induces on that column the arrow $(0,i)\to(1,j)$; also, a contains the arrow from $(0,0)$ on the left column to $(0,0)$ on the right. Symmetrically, each $(B,\text{R}) \in \Delta_h$ maps to a $b \in \Gamma_{2h}$ whose left column encodes B, now using level 1 for origins and level 0 for destinations: each arrow $j\leftarrow i \in B$ induces on that column the arrow $(1,i)\to(0,j)$; also, b contains an arrow from $(1,i)$ on the left column to $(1,i)$ on the right, for each i such that $0\leftarrow i \in B$. (See Fig. 5 for an example.)

Easily, each k-long cycle witnessing that A, B match through 0 induces in ab a k-long cycle which starts at vertex $(0,0)$ of the middle column, alternates both between the two symbols and between the two levels, and returns to $(0,0)$ via an arrow of b of the form $(1,i)\to(0,0)$; hence B contains the arrow $0\leftarrow i$, and thus b contains the arrow from $(1,i)$ on the left to $(1,i)$ on the right; overall, ab contains the live path which starts on $(0,0)$ on the leftmost column, moves to $(0,0)$ on the middle column, and continues as in the cycle, diverging from it only in the last step from $(1,i)$, to jump right. Conversely, if ab is live, then each live path is of this form, and thus cannot exist without a respective cycle through $(0,0)$ on the middle column; this cycle induces a cycle in the overlay of (A,L) and (B,R), through 0 of the left column. So, A, B match through 0 iff ab is live. □

Lemma 9 2D $\ni$ REL MATCH *iff* REL MATCH $\leq_{\text{h}}$ FUN MATCH.

Proof The backward direction follows from Lemma 4 and Corollary 3. For the forward direction, suppose RM_h is solved by an s-state 2DFA M, where $s = \text{poly}(h)$. We give a homomorphism $f : \Delta_h \cup \{\vdash,\dashv\} \to (\Delta'_s)^*$ which maps $\vdash$ and $\dashv$ to ϵ, and every (A,L) or (B,R) to a symbol (α,L) or (β,R) respectively, such that the relations A, B match iff the functions α, β *match through* 0. This proves $\text{RM}_h \leq_\text{h} \text{FZM}_s$, which implies $\text{RM}_h \leq_\text{h} \text{FM}_{2s^2}$ (by the first half of Lemma 7.2). Since $2s^2 = \text{poly}(h)$, it follows that REL MATCH $\leq_\text{h}$ FUN MATCH.

To build f, we assume that $M = ([s], \Delta_h, \delta, 0, s{-}1)$ and that the only transition into state 0 is the self-loop $\delta(0,\vdash) = (0,\text{R})$. (This does not harm generality: to bring M into the form $([s], \Delta_h, \delta, 0, s{-}1)$, we simply rename its states appropriately; to guarantee the assumption for the start state, we simply introduce a fresh start state $0'$ and let $\delta(0',\vdash) := (0',\text{R})$ and $\delta(0', a) := \delta(0, a)$ for all $a \in \Delta_h \cup \{\dashv\}$, also incorporating the slight increase in size into the assumption that s is polynomial in h.)

Then, for every relation A, we set $f(A,\text{L}) := (\alpha,\text{L})$ where $\alpha : [s] \rightharpoonup [s]$ satisfies $\alpha(p) = q$ iff the computation of M from state p on the right symbol of $\vdash(A,\text{L})$ falls off the rightmost boundary into state q. Symmetrically, for every B, we set $f(B,\text{R}) := (\beta,\text{R})$ where $\beta : [s] \rightharpoonup [s]$ satisifies $\beta(p) = q$ iff *either* the computation of M from p on the left symbol of $(B,\text{R})\dashv$ falls off the leftmost boundary into q (hence $q \neq 0$, by our assumption for 0) *or* $q = 0$ and the computation falls off $\dashv$ into $s{-}1$. We claim that A, B match iff α, β match through 0.

For the forward direction, suppose A, B match. Then the computation $c = \big((q_t, j_t)\big)_{0 \leq t \leq m}$ of M on $\vdash(A,\text{L})(B,\text{R})\dashv$ is accepting. By our assumptions for M, we know M starts in state 0 on $\vdash$, moves right into state 0 again, and eventually falls off $\dashv$ into $s{-}1$. So, c has the form $(0,0), (0,1), \ldots, (s{-}1, 4)$. Let us drop the first configuration, and 'split' the remaining sequence $(0,1), \ldots, (s{-}1, 4)$ every time it crosses the middle boundary of the input (between cells 1 and 2):

- at forward crossings, the sequence looks like $\ldots, (p,1), (q,2), \ldots$ and the 'split' produces a part ending in $(p,1), (q,2)$ and a part starting with $(q,2)$;
- at backward crossings, the sequence looks like $\ldots, (p,2), (q,1), \ldots$ and the 'split' produces a part ending in $(p,2), (q,1)$ and a part starting with $(q,1)$.

The result is a list of subsequences $c_1, c_2, \ldots, c_k$, where k is even, every odd-indexed c_i computes on $\vdash(A,\text{L})$, and every even-indexed c_i computes on $(B,\text{R})\dashv$. Moreover, if we let q_i be the state of the first configuration of c_i, then:

- $q_1 = 0$;
- every odd-indexed c_i has the form $(q_i, 1), \ldots, (q_{i+1}, 2)$, and thus proves $\alpha(q_i) = q_{i+1}$;
- every even-indexed c_i before c_k is of the form $(q_i, 2), \ldots, (q_{i+1}, 1)$, and thus proves $\beta(q_i) = q_{i+1}$; and c_k is of the form $(q_k, 2), \ldots, (s{-}1, 4)$, and thus proves $\beta(q_k) = 0$.

Overall, the path

$$0 = q_1 \xrightarrow{\alpha} q_2 \xrightarrow{\beta} q_3 \xrightarrow{\alpha} q_4 \xrightarrow{\beta} \cdots \xrightarrow{\alpha} q_k \xrightarrow{\beta} 0$$

is a cycle through 0 on the left column of the overlay. Hence α, β match through 0.

Conversely, suppose α, β match through 0. If $q_1, q_2, \ldots, q_k \in [s]$ is a cycle that proves this, then:

- k is even;
- $q_1 = 0$;
- every odd-indexed q_i satisfies $\alpha(q_i) = q_{i+1}$, proving that M from q_i on the right symbol of $\vdash(A,\text{L})$ falls off the right boundary into q_{i+1};
- every even-indexed q_i before q_k satisfies $\beta(q_i) = q_{i+1}$, proving that M from q_i on the left symbol of $(B,\text{R})\dashv$ falls off the left boundary into q_{i+1};
- q_k satisfies $\beta(q_k) = 0$, proving that M from q_k on the left symbol of $(B,\text{R})\dashv$ falls off $\dashv$ into $s{-}1$ (here we use the assumption that no transition enters state 0 on a symbol of Δ_h, hence $\beta(q_k) = 0$ cannot be due to a computation that falls off the left boundary).

Clearly then, the 'concatenation' of these computation segments proves that M from state 0 on cell 1 of $\vdash(A,\text{L})(B,\text{R})\dashv$ falls off $\dashv$ into $s{-}1$. Given that, in addition, M performs the rightward step from state 0 on $\vdash$ onto cell 1 and state 0 again, we see that M indeed accepts this input. Hence, A, B match. □

Combining Lemma 8 and Lemma 9, we see that Conjectures A and B are indeed equivalent.

6 Conclusion

We proved several characterizations of $\mathsf{L/poly}$ versus NL in terms of unary, binary, or general 2FAs. Our main theorem complements two recent improvements [5,8] of an old theorem [1], and our approach introduced some of the concepts and tools that had been asked for in [7] for enriching the framework of [10].

It would be interesting to see similar characterizations in terms of unary 2FAs for the uniform variant of the question (L versus NL), or for variants for other bounds for space and advice length (e.g., $\mathsf{LL/polylog}$ versus NLL [8]). Another interesting direction is to elaborate further on the comparison between 2FA computations on short and on unary inputs; for example, using the ideas of this article, one can show that for every $\mathcal{L} \in \mathsf{2N/poly}$ there is $\mathcal{L}' \in \mathsf{2N/unary}$ such that $\mathcal{L} \preceq_{\mathsf{2D}}^{\text{lac}} \mathcal{L}'$. Finally, further work is needed to fully understand the capabilities and limitations of the reductions introduced in this article.

Acknowledgements The first author has been supported by a Marie Curie Intra-European Fellowship (PIEF-GA-2009-253368) within the European Union Seventh Framework Programme (FP7/2007-2013).

References

1. 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)

2. Geffert, V., Guillon, B., Pighizzini, G.: Two-way automata making choices only at the endmarkers. In: Proceedings of LATA, pp. 264–276 (2012)
3. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theoretical Computer Science **295**, 189–203 (2003)
4. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. Information and Computation **205**(8), 1173–1187 (2007)
5. Geffert, V., Pighizzini, G.: Two-way unary automata versus logarithmic space. Information and Computation **209**(7), 1016–1025 (2011)
6. Hardy, G.H., Wright, E.M.: An introduction to the theory of numbers. Oxford Science Publications (1979)
7. Kapoutsis, C.: Size complexity of two-way finite automata. In: Proceedings of DLT, pp. 47–66 (2009)
8. Kapoutsis, C.: Two-way automata versus logarithmic space. In: Proceedings of CSR, pp. 197–208 (2011)
9. Kunc, M., Okhotin, A.: Describing periodicity in two-way deterministic finite automata using transformation semigroups. In: Proceedings of DLT, pp. 324–336 (2011)
10. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proceedings of STOC, pp. 275–286 (1978)
11. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences **4**, 177–192 (1970)