Two-way automata characterizations of L/poly versus NL

Christos A. Kapoutsis[1,⋆] and Giovanni Pighizzini[2]

[1] LIAFA, Université Paris VII, France
[2] DICo, Università degli Studi di Milano, Italia

Abstract. Let $\mathsf{L/poly}$ 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 $\mathsf{L/poly} \supseteq \mathsf{NL}$ 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 $\mathsf{L/poly} \not\supseteq \mathsf{NL}$.

1 Introduction

A prominent open question in complexity theory asks whether nondeterminism is essential in logarithmic-space Turing machines. Formally, this is the question 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 [7] proved that $\mathsf{L/poly} \supseteq \mathsf{NL}$ *iff for every* s*-state* σ*-symbol* 2NFA *there is a* poly(s)*-state* 2DFA

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

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.

A natural question arising from these developments is whether the theorem of [5] can be strengthened to resemble that of [7]: 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* $\mathrm{poly}(s)$*-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 [7]), 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, 8] and a special unary encoding for graphs.

We work in the complexity-theoretic framework of [9]. We focus on the class 2D of (families of promise) problems solvable by polynomial-size 2DFAs, and on the corresponding classes $\mathsf{2N/poly}$ and $\mathsf{2N/unary}$ 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 [7] 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 [7] and the equivalence $\mathsf{2D} \supseteq \mathsf{2N/poly} \Leftrightarrow \mathsf{2D} \supseteq \mathsf{2N/unary}$, obtained by our analysis of the common properties or 'short' and unary inputs.

Our approach returns several by-products of independent interest, already anticipated in [6] for enriching the framework of [9]: 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 $\mathsf{2N/poly}$ and $\mathsf{2N/unary}$, respectively, under such reductions (Cor. 2); the closure of 2D under such reductions (Cor. 3); and the realization of the central role of 2OFAs in this framework (as also recognized in [2]). In the end, our main theorem (Th. 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 $\mathsf{2O}$ is the analogue of $\mathsf{2D}$ 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 $h \geq 1$, 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. If x is a string, then $|x|$, x_i, and x^i are its length, its i-th symbol ($1 \leq i \leq |x|$), and the concatenation of i copies of it ($i \geq 0$). The empty string is denoted by ϵ.

The *binary encoding* of a subgraph G of K_h, 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 $\mathtt{1}$. 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 $\mathtt{1}$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 $\delta \subseteq S \times (\Sigma \cup \{\vdash,\dashv\}) \times S \times \{\text{L},\text{R}\}$, 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 say 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 $(q, a, \,.\,, \,.\,)$; if this holds also for each $(q, a) \in S \times \{\vdash,\dashv\}$, then M is *deterministic* (2DFA). We say 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. We say 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

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

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 $x_i = \dashv$ & $p = q_f$ & $d = \text{R}$, in which case it just falls off $\dashv$ into q'_f.

If M is unary, then it can be simulated by a SOFA $\tilde{M}$ with $\tilde{s} = O(s^2)$ states [3]. This can then be converted into a ROFA M' with $2\tilde{s} + 1$ states, which simply replaces backward sweeps with forward ones—a straightforward simulation, since reversing a unary input does not change it. Hence, M' is also of size $O(s^2)$. □

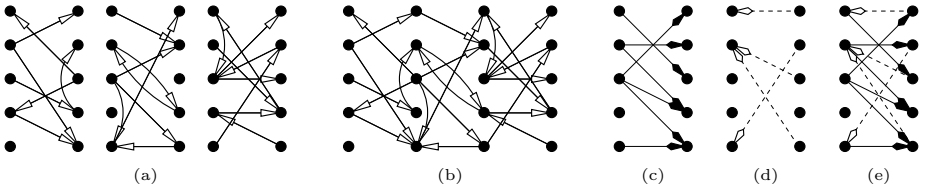

Fig. 1. (a) Three symbols of Γ_5. (b) Their string as a multi-column graph. (c) A symbol $(A,\text{L}) \in \Delta_5$. (d) A symbol $(B,\text{R}) \in \Delta'_5$. (e) The overlay of the symbols of (c) and (d).

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* $\mathsf{2N/unary} = \mathsf{RO/unary}$.

2.2 Problems

The *graph accessibility problem* on h vertices is: "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:

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

and the corresponding families $\text{BGAP}{:=}(\text{BGAP}_h)_{h\geq 1}$ and $\text{UGAP}{:=}(\text{UGAP}_h)_{h\geq 1}$.

Lemma 2. BGAP_h *and* UGAP_h *are solved by* ROFAs *with* $O(h^3)$ *and* $O(h^4 \log h)$ *states, respectively. Hence* $\text{BGAP} \in \mathsf{RO/poly}$ *and* $\text{UGAP} \in \mathsf{RO/unary}$.

The *two-way liveness* problem on height h, defined over the alphabet $\Gamma_h := \mathcal{P}(([h]\times\{\text{L,R}\})^2)$ of all h-tall directed two-column graphs (Fig. 1a), is: "Given a string x of such graphs, 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. 1b) 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 and families

$$\textsc{short twl}_h := \big(\{x \in \Gamma_h^{\leq h} \mid x \text{ is live}\},\ \{x \in \Gamma_h^{\leq h} \mid x \text{ is dead}\} \big)$$
$$\textsc{compact twl}_h := \big(\{ab \in \Gamma_h^{2} \mid ab \text{ is live}\},\ \{ab \in \Gamma_h^{2} \mid ab \text{ is dead}\} \big)$$

and $\textsc{short twl}:=(\textsc{short twl}_h)_{h\geq 1}$, $\textsc{compact twl}:=(\textsc{compact twl}_h)_{h\geq 1}$.

Lemma 3. $\textsc{twl}_h$ *is solved by a* 2NFA *with* $2h$ *states. Hence* SHORT TWL *and* COMPACT TWL *are both in* RO/poly.

The *relational match* problem on $[h]$ is defined over the alphabet $\Delta_h := \mathbb{P}([h]\times[h])\times\{\text{L},\text{R}\}$ 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. 1c); a symbol (B,R) denotes a similar graph with leftward arrows chosen by B (Fig. 1d). If the overlay of these two graphs (Fig. 1e) contains a cycle, we say that the symbols *match*, or just that A, B *match*. We consider the problem

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

of checking that two given relations match, and set $\textsc{rel match}:=(\textsc{rm}_h)_{h\geq 1}$. We also let $\textsc{fm}_h$ be the restriction of $\textsc{rm}_h$ to the alphabet $\Delta'_h := ([h]\rightharpoonup[h])\times\{\text{L},\text{R}\}$, where all relations are partial functions, and set $\textsc{fun match}:=(\textsc{fm}_h)_{h\geq 1}$.

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

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 $T = (S, \Sigma, \Gamma, \delta, q_0, q_\text{f})$ of a set of states S, an input alphabet Σ, an output alphabet Γ, a start state $q_0 \in S$, a final state $q_\text{f} \in S$, and a transition function $\delta : S \times (\Sigma \cup \{\vdash,\dashv\}) \to S \times \{\text{L},\text{R}\} \times \Gamma^*$, 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. 3) 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 function $f : \Sigma^* \to (\Sigma')^*$ is a (*mapping*) *reduction* of $\mathfrak{L}$ to $\mathfrak{L}'$ if it is computable by a 2DFT and every $x \in \Sigma^*$ satisfies $x \in L \Rightarrow f(x) \in L'$ and $x \in \tilde{L} \Rightarrow f(x) \in \tilde{L}'$. If $\Sigma' = \{\mathtt{0}\}$ and f is prime-computable by a 2DFT, then f is a *prime reduction*. If $f(x)$ is always just $g(\vdash)g(x_1)\cdots g(x_{|x|})g(\dashv)$ for some homomorphism $g : \Sigma \cup \{\vdash,\dashv\} \to (\Sigma')^*$, then f is a *homomorphic reduction*. 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 $\mathfrak{L} \leq_{\mathrm{m}} \mathrm{BGAP}_{2s+1}$ and $\mathfrak{L} \preceq_{\mathrm{m}} \mathrm{UGAP}_{2s+1}$ 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 $|f(x)| = |x|{+}2$ (e.g., see [7, 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. First, we remove from M all transitions performed on $\vdash$ or $\dashv$. Second, 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'). Third, 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 $\langle G(x)\rangle_1$ (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 prime powers 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. The members of $\mathcal{T}$ are *laconic* if every T_h performs $\leq p(h)$ printing steps on each input, for some polynomial p. 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; if the problems in $\mathcal{L}'$ are unary and $\mathcal{T}$ prime-computes $\mathcal{F}$, then $\mathcal{L}$ *prime-reduces to* $\mathcal{L}'$ *in polynomial size* ($\mathcal{L} \preceq_{\mathsf{2D}} \mathcal{L}'$); if the 2DFTs in $\mathcal{T}$ are laconic, then $\mathcal{L}$ (*prime-*) *reduces to* $\mathcal{L}'$ *in polynomial size/print* ($\mathcal{L} \leq^{\mathrm{lac}}_{\mathsf{2D}} \mathcal{L}'$ or $\mathcal{L} \preceq^{\mathrm{lac}}_{\mathsf{2D}} \mathcal{L}'$). If every $\mathfrak{L}_h$ homomorphically reduces to $\mathfrak{L}'_{p(h)}$ for some polynomial p, then $\mathcal{L}$ *homomorphically reduces to* $\mathcal{L}'$ ($\mathcal{L} \leq_{\mathrm{h}} \mathcal{L}'$); if the underlying homomorphisms are tight, then $\mathcal{L}$ *reduces to* $\mathcal{L}'$ *under tight homomorphisms* ($\mathcal{L} \leq^{\mathrm{t}}_{\mathrm{h}} \mathcal{L}'$).

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

4 Closures

We now prove that $\mathsf{2D}$ 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* $\mathfrak{L}_1 \leq_{\mathrm{m}} \mathfrak{L}_2$ *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* $\mathfrak{L}_1 \preceq_{\mathrm{m}} \mathfrak{L}_2$ *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* $\mathfrak{L}_1 \leq_{\mathrm{h}} \mathfrak{L}_2$*, then* $\mathfrak{L}_1$ *is solved by a* 2DFA *with* $2s$ *states.*

Proof. Part 3 is well-known (e.g., see [7, 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.

Easily, M_1 can implement this algorithm with $S_1 := S_2 \times \{\mathrm{L},\mathrm{R}\} \times [t] \times S \times [t]$, where state (q, d, i, p, j) 'means' 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 [8, 3], it follows that there exist a length bound $l = O(s)$ and a $O(s)$-state RDFA $\tilde{M}_2 = (\tilde{S}_2, \{\mathtt{0}\}, \tilde{\delta}, \tilde{q}_0, \tilde{q}_\mathrm{f})$ such that $\tilde{M}_2$ agrees with M_2 on all lengths $\geq l$, and is in the following 'normal form'. The states $\tilde{S}_2 \setminus \{\tilde{q}_0{,}\tilde{q}_\mathrm{f}\}$ 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 $\tilde{q}_\mathrm{f}$ 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 $n = |T(x)|$ 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 $n = \hat{n}$, 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. 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.

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 $\mathtt{\#}z_i$ of $T(x)$, then $n \bmod l_j$ equals

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

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 $\mathtt{\#}z_i$ inside a string printed by T. 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.

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

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

5 Characterizations and conjectures

Our main theorem is now a direct consequence of [7] 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 [7]; (1)⇔(2)⇔(5), by Cor. 2.1 and the closure of 2D under polynomial-size/print reductions; (3)⇔(4)⇔(5) by Cor. 2.2 and the closure of 2D under polynomial-size prime reductions; and (1)⇔(6) by Cor. 2.3 and the closure of 2D under homomorphic reductions. □

The statements of Th. 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) 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. 6).

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

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

Lemma 7. *The following reductions hold:*

1. REL ZERO-MATCH $\leq_{\text{h}}$ REL MATCH *and* REL MATCH $\leq_{\text{h}}$ REL ZERO-MATCH*.*

2. FUN ZERO-MATCH $\leq_{\text{h}}$ FUN MATCH *and* FUN MATCH $\leq_{\text{h}}$ FUN ZERO-MATCH*.*

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

Lemma 8. *The following statements hold:*
1. COMPACT TWL $\leq_{\mathsf{2D}}^{\mathrm{lac}}$ REL MATCH *and* REL MATCH $\leq_{\mathrm{h}}$ COMPACT TWL.
2. $\mathsf{2D} \ni$ REL MATCH *iff* REL MATCH $\leq_{\mathrm{h}}$ FUN MATCH.

Combining the two facts, 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, 7] of an old theorem [1], and our approach introduced some of the concepts and tools that had been asked for in [6] for enriching the framework of [9].

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 [7]). 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 $\mathcal{L} \preceq_{\mathsf{2D}}^{\mathrm{lac}} \mathcal{L}'$. Finally, further work is needed to fully understand the capabilities and limitations of the reductions introduced in this article.

References

1. P. Berman and A. Lingas. On complexity of regular languages in terms of finite automata. Report 304, Institute of Computer Science, Polish Academy of Sciences, Warsaw, 1977.
2. V. Geffert, B. Guillon, and G. Pighizzini. Two-way automata making choices only at the endmarkers. In *Proceedings of LATA*, 2012. To appear.
3. V. Geffert, C. Mereghetti, and G. Pighizzini. Converting two-way nondeterministic unary automata into simpler automata. *Theoretical Computer Science*, 295:189–203, 2003.
4. V. Geffert, C. Mereghetti, and G. Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007.
5. V. Geffert and G. Pighizzini. Two-way unary automata versus logarithmic space. *Information and Computation*, 209(7):1016–1025, 2011.
6. C. Kapoutsis. Size complexity of two-way finite automata. In *Proceedings of DLT*, pages 47–66, 2009.
7. C. Kapoutsis. Two-way automata versus logarithmic space. In *Proceedings of CSR*, pages 197–208, 2011.
8. M. Kunc and A. Okhotin. Describing periodicity in two-way deterministic finite automata using transformation semigroups. In *Proceedings of DLT*, pages 324–336, 2011.
9. W. J. Sakoda and M. Sipser. Nondeterminism and the size of two-way finite automata. In *Proceedings of STOC*, pages 275–286, 1978.