Nondeterminism is essential in small two-way finite automata with few reversals

Christos A. Kapoutsis

Laboratoire d'Informatique Algorithmique: Fondements et Applications
Université Paris VII

Abstract

On every n-long input, every two-way finite automaton (2FA) can reverse its input head $O(n)$ times before halting. A 2FA *with few reversals* is an automaton where this number is only $o(n)$. For every h, we exhibit a language that can be recognized by an h-state nondeterministic 2FA with few reversals, but requires $\Omega(2^h)$ states on every deterministic 2FA with few reversals.

1. Introduction

A long-standing open question in the theory of computation, posed already in the seventies [1, 2], is whether every two-way nondeterministic finite automaton (2NFA) has an equivalent deterministic one (2DFA) with at most polynomially more states.

The answer to this question is conjectured to be negative. Indeed, this has been confirmed in several special cases: for automata that are *single-pass* (i.e., they halt upon reaching an endmarker [1]) or *sweeping* (i.e., they reverse their input head only on endmarkers [3, 4]) or *almost oblivious* (i.e., they exhibit $o(n)$ distinct input head trajectories over all n-long inputs [5]) or *moles* (i.e., they explore the configuration graph implied by the input [6]). For *unary* automata, however, a non-trivial upper bound has been established: every unary 2NFA admits a deterministic simulator with only quasi-polynomially more states [7]. It is also known that the final answer to this question, both for general and for unary alphabet, will have implications for the old question whether nondeterminism is essential in space-bounded Turing machines [8, 9, 10].

Here we confirm the general conjecture in yet another special case: for automata that reverse their input head (anywhere on the tape, but) only $o(n)$ times on every n-long input before halting. These '2FAs *with few reversals*' stand naturally between sweeping 2FAs, which perform only $O(1)$ reversals and only on the endmarkers, and general 2FAs, which perform $O(n)$ reversals (cf. Fact 3).

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

Email address: christos.kapoutsis@liafa.jussieu.fr (Christos A. Kapoutsis)

Theorem 1. *For every h, there is a language that requires $\Omega(2^h)$ states on every* 2DFA with few reversals *but only h states on a* 2NFA with few reversals.

The family of languages witnessing this theorem is ONE-WAY LIVENESS [2] (as in several other theorems [3, 5, 6]) and the promised h-state 2NFAs are actually one-way (that is, their 'few' reversals are in fact 'zero'). Hence, the theorem can be seen as a generalization of the main theorem of [3], which states that ONE-WAY LIVENESS requires exponentially many states on sweeping 2DFAs.

Given Theorem 1, two natural questions arise. First, *does the theorem really generalize the one of* [3], or can it perhaps follow from it by proving that the gap from few-reversal 2DFAs to sweeping 2DFAs is only polynomial? Second, *does the full conjecture really generalize the theorem*, or can it perhaps follow from it by proving that the gap from general 2DFAs to few-reversal 2DFAs is only polynomial? We provide affirmative answers to both of these questions.

Theorem 2. *For every h, there is a language that requires $2^{\Omega(h)}$ states on every* sweeping 2DFA *but only $O(h)$ states on a* 2DFA with 2 reversals.

Theorem 3. *For every h, there is a language that requires $\Omega(2^h)$ states on every* 2DFA with few reversals *but only $O(h^2)$ states on a* general 2DFA.

We note that Theorems 1, 2, and 3 together can been seen as full answers to Research Problems 2 and 3 proposed by J. Hromkovič in 2002 [11].

We consider the second half of Theorem 1 (i.e., the upper bound) known, and leave the proofs of Theorems 2 and 3 for the final sections (Sections 7 and 6, respectively). We thus devote most of this article to proving the first part of Theorem 1 (i.e., the lower bound). Our argument is structured similarly to the one in [3]: after fixing terminology, notation, and strategy (Section 2), we introduce *generic strings* (Section 3), study the *blocks* defined by them (Section 4), use such blocks to build a family of *hard instances* (Section 5.1), and conclude by applying the linear algebra bound on a family of vectors derived from these instances (Section 5.2).

2. Preparation

2.1. Functions

Let A, B, C be sets, and $f : A \rightharpoonup B$ and $g : B \rightharpoonup C$ be partial functions. We write $\overline{A}$ and $\mathbb{P}(A)$ for the complement and for the powerset of A. By $\mathrm{id}_A : A \to A$ we denote the identity function on A. The image of a set $X \subseteq A$ under f is $f[X] := \{f(a) \mid a \in X \text{ and } f(a) \text{ is defined}\}$. The pre-image of an element $b \in B$ under f is $f^{-1}(b) := \{a \in A \mid f(a) = b\}$. The composition of f and g is the partial function $f \circ g : A \rightharpoonup C$ which is defined on an element $a \in A$ iff both $f(a)$ and $g(f(a))$ are defined, and then $(f \circ g)(a) = g(f(a))$. If $A = B$, then the k-fold composition of f with itself is denoted by f^k. By $f \leq g$ we mean that, whenever $f(a)$ is defined, $g(a)$ is also defined and equals $f(a)$. Easily, $\leq$ is a partial order on the set $\{f \mid f : A \rightharpoonup A\}$ of all partial functions on A, and every total function is a maximal element in this order.

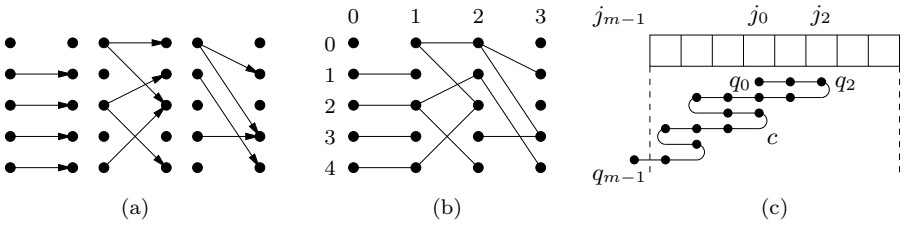

Figure 1: (a) Three symbols in Σ_5; e.g., the third symbol is $\{(0,1),(0,3),(1,4),(3,3)\}$. (b) The string defined by the three symbols on the left, simplified and indexed; here $n = 3$ and $\xi = \{(2,4)\}$. (c) A left-hitting computation c with $m = 15$ and $r(c) = 5$; points 2, 8, and 12 are backward reversals, while points 6 and 11 are forward reversals.

Fact 1. *For all $f, f_1, f_2 : A \rightharpoonup A$ we have $f_1 \leq f_2 \Longrightarrow f \circ f_1 \leq f \circ f_2$.*

Proof. Let $f_1 \leq f_2$ and pick any $a \in A$ for which $(f \circ f_1)(a)$ is defined. Then $f(a)$ is defined and equals some $b \in A$, and $f_1(b)$ is defined and equals some $c \in A$, and $c = (f \circ f_1)(a)$. By $f_1 \leq f_2$ and $f_1(b) = c$, we know $f_2(b)$ is also defined, and equals c. By $f(a) = b$ and $f_2(b) = c$, we know $(f \circ f_2)(a)$ is also defined, and equals c. □

2.2. Problems

Let Σ be an alphabet. If $z \in \Sigma^*$ is a string, we write z^{R}, $|z|$, and z_j for its reverse, length, and j-th symbol (if $1 \leq j \leq |z|$). If $Z \subseteq \Sigma^*$, then the corresponding set of reverses is $Z^{\mathsf{R}} := \{z^{\mathsf{R}} \mid z \in Z\}$.

A (*promise*) *problem* over Σ is any pair $\mathfrak{L} = (L, \tilde{L})$ of disjoint subsets of Σ^*. Its *positive instances* are all $w \in L$, whereas all $w \in \tilde{L}$ are its *negative instances*. The set $L \cup \tilde{L}$ of all instances is the *promise*. A machine *solves* $\mathfrak{L}$ if it accepts every positive instance but no negative one. If $\tilde{L} = \overline{L}$, then $\mathfrak{L}$ is a *language*.

Let $h \geq 1$ and $[h] := \{0, \ldots, h-1\}$. The alphabet $\Sigma_h := \mathbb{P}([h] \times [h])$ consists of all two-column directed graphs with h nodes per column and only rightward arrows (Fig. 1a). An n-long string $z \in \Sigma_h^*$ is naturally viewed as an $(n+1)$-column graph, where every arrow connects successive columns (Fig. 1b); we usually index the columns from 0 to n and, for simplicity, drop the directions of the arrows. The *connectivity* of z is the set of pairs

$$\xi(z) := \{\ (a,b) \in [h] \times [h] \ \mid \text{ there exists a path (of length } n) \text{ from node } a \text{ of column 0 to node } b \text{ of column } n\ \}.$$

If $\xi(z) \neq \emptyset$ then z contains paths from the leftmost to the rightmost column, and we say that it is *live*; otherwise we call it *dead*. The language

$$\text{OWL}_h = \text{ONE-WAY LIVENESS}_h := \{\ z \in \Sigma_h^* \ \mid \ \xi(z) \neq \emptyset\ \}$$

represents the computational task of checking that a string is live [2].

2.3. Machines

A *two-way deterministic finite automaton* (2DFA) is any tuple of the form $M = (Q, \Sigma, \delta, q_s, q_a, q_r)$, where Q is a set of *states*, Σ is an *alphabet*, $q_s, q_a, q_r \in Q$ are respectively the *start*, *accept*, and *reject* states, and

$$\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{\mathtt{l}, \mathtt{r}\}$$

is the (total) *transition function*, for $\vdash, \dashv \notin \Sigma$ the left and right endmarkers, and $\mathtt{l}, \mathtt{r}$ the left and right directions. An input $w \in \Sigma^*$ is presented to M surrounded by the endmarkers, as $\vdash w \dashv$. The computation starts at q_s and on $\vdash$. In each step, the next state and head move are derived from δ and the current state and symbol. Endmarkers are never violated, except for $\dashv$ when the next state is q_a or q_r; that is, $\delta(\cdot, \vdash)$ is always of the form $(\cdot, \mathtt{r})$, whereas $\delta(\cdot, \dashv)$ is always $(q_a, \mathtt{r})$ or $(q_r, \mathtt{r})$ or of the form $(\cdot, \mathtt{l})$. Hence, the computation either loops, or falls off $\dashv$ into q_r, or falls off $\dashv$ into q_a. In the last case, we say that M *accepts* w.

In general, the *computation of M from state p on the j-th symbol of string z*, denoted by $\textsc{comp}_{M,p,j}(z)$, is the longest sequence

$$c = ((q_t, j_t))_{0 \le t < m}$$

such that $0 < m \le \infty$, $(q_0, j_0) = (p, j)$, and every next (q_t, j_t) is derived from the previous one via δ and z in the natural way. We call (q_t, j_t) the *t-th point* of the computation, and m the *length*. If $m = \infty$ then c *loops*; otherwise, $j_{m-1} = 0$ or $|z|+1$ and c *hits left* or *hits right*, respectively, *into* q_{m-1} (Fig. 1c). We say that a computation $c' = ((q'_t, j'_t))_{0 \le t < m'}$ *parallels* c if it is a 'shifted copy' of c, in the sense that $m' = m$ and $q'_t = q_t$ and $j'_t = j_t + j_*$ for some j_* and all t.

The L-*computation of M from p on z* is the computation

$$\textsc{lcomp}_{M,p}(z) := \textsc{comp}_{M,p,1}(z)$$

and is called a LR-*traversal*, a L-*turn*, or a L-*loop*, depending on whether it hits right, hits left, or loops. Similarly, the R-*computation of M from p on z*

$$\textsc{rcomp}_{M,p}(z) := \textsc{comp}_{M,p,|z|}(z)$$

is a RL-*traversal*, a R-*turn*, or a R-*loop*. Two L-/R-computations *resemble* each other if they share the same first state, the same type (L/R-turn/loop, LR/RL-traversal), and the same last state (when it exists). The (*full*) *computation of M on $w \in \Sigma^*$* is the computation

$$\textsc{comp}_M(w) := \textsc{lcomp}_{M,q_s}(\vdash w \dashv)\,.$$

Hence, M accepts w iff $\textsc{comp}_M(w)$ hits right into q_a.

Given a string $w = uzv$, the *decomposition of $c := \textsc{comp}_M(w)$ relative to z* is the unique sequence $c_0, c_1, \ldots$ of computations, called *segments*, which is derived by splitting c wherever it enters or exits z. More precisely, for each point (q_t, j_t) of c produced by a step that crosses the u-z or the z-v boundary, we replace

(q_t, j_t) by two copies of itself and then split c between these two copies. Note that every segment c_i for even i is a computation on $\vdash u$ or on $v\dashv$, whereas every c_i for odd i is a computation on z. Moreover, c halts iff there exists a last segment c_m and m is even and c_m falls off $\dashv$.

We say that M is *nondeterministic* (a 2NFA) if δ maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to the *powerset* of $Q \times \{\mathtt{l}, \mathtt{r}\}$. Then every $\textsc{comp}_{M,p,j}(z)$ is a *set* of computations, and M accepts w iff some $c \in \textsc{comp}_M(w)$ hits right into q_a.

A *reversal* in a computation c is any point (q_t, j_t) whose predecessor and successor exist and lie on the same side with respect to the point (Fig. 1c). More carefully, (q_t, j_t) is a reversal if $t \neq 0, m{-}1$ and either $j_{t-1}, j_{t+1} < j_t$ (*backward* reversal) or $j_t < j_{t-1}, j_{t+1}$ (*forward* reversal). We write $r(c)$ for the total number of reversals in c. Note that $0 \leq r(c) \leq \infty$ and the following holds.

Fact 2. *For every computation c, we have $r(c) = \infty$ iff c is looping.*

For every length $n \geq 0$, we write $r_M(n)$ for the maximum $r(c)$ over all full computations c of M on n-long inputs. When finite, this is at most linear.

Fact 3. *For every s-state* 2DFA *M and every length n, either $r_M(n) = \infty$ or $r_M(n)$ is even and at most $(s-1)(n+2)$.*

Proof. If M loops on any of its n-long inputs, then clearly $r_M(n) = \infty$. Otherwise, for any n-long input w, we know $c := \textsc{comp}_M(w)$ is halting. Hence, c starts on $\vdash$ and eventually falls off $\dashv$. Therefore, every backward reversal is followed by a forward one. So, $r(c)$ is even. Moreover, *on each of the $n{+}2$ tape cells, c performs at most $s{-}1$ reversals.* Thus, $r(c) \leq (s-1)(n+2)$. Since w was arbitrary, we conclude that $r_M(n)$ is also even and at most $(s-1)(n+2)$.

It remains, of course, to prove the italicised claim above.

We start by noting that *on every cell, at least one point of c is not a reversal.* Indeed, if that is not the case, then there exists a cell on which all points are reversals. The first of these reversals is a backward one, because it is the first visit to the cell and thus came from the left. The second reversal is also a backward one, because it is the second visit to the cell and, since the first visit was a backward reversal, must have come from the left, too. And so on. Hence, all points on the cell are backward reversals, which means c never moves past this cell. So, c never falls off $\dashv$ —a contradiction.

Now suppose there is a cell on which c performs more than $s{-}1$ reversals. Since every reversal is a point and at least one of the points on that cell is not a reversal, c has more than s points there. Hence, two of them use the same state, and are thus identical. So, c repeats a point. So, it loops —a contradiction. □

If M performs all its reversals on the endmarkers, we call it *sweeping* (a SDFA or SNFA). If it performs no reversals, we call it *one-way* (a 1DFA or 1NFA).

2.4. Building hard instances

Hard instances for 2DFAs are built in three stages. We start with *generic strings*, which buy us some basic stability in the machine's behavior. We then

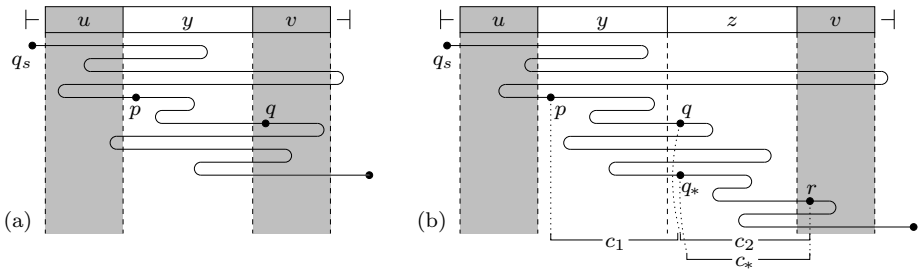

Figure 2: (a) Computing on uyv. The segment from p to q parallels $\textsc{lcomp}_{M,p}(y)$, but differs from it in that every point's position is incremented by $|{\vdash}u|$. (b) Computing on $uyzv$.

use generic strings to build *blocks*, on which we draw a set of requirements for how the machine must compute. In order to prove that the machine must indeed meet these requirements, we iterate the blocks into *much longer strings* and use the fact that the machine decides correctly there. Finally, we argue that meeting the requirements on the blocks is impossible for any machine with subexponentially many states. This general strategy originates in [3]; its instantiation here for 2DFAs improves on the analysis of [6, §3].

For the rest of the article, we fix a 2DFA $M{=}(Q, \Sigma, \delta, q_s, q_a, q_r)$ and drop 'M' from all subscripts. For example, $\textsc{lcomp}_{M,p}(w)$ and $r_M(n)$ will be denoted by just $\textsc{lcomp}_p(w)$ and $r(n)$.

3. Generic strings

For each $y \in \Sigma^*$, consider all states that can be produced by LR-traversals of y inside full computations of M (Fig. 2a), called the LR-*outcomes* of y:

$$Q_{\text{LR}}(y) := \big\{q \in Q \mid \text{there exist } p \text{ and } u, v \text{ such that } \textsc{lcomp}_p(y) \text{ appears in } \textsc{comp}(uyv) \;\&\; \text{hits right into } q\big\}, \quad (1)$$

where a computation on y 'appears in $\textsc{comp}(uyv)$' if it parallels one of the odd-indexed segments in the decomposition of $\textsc{comp}(uyv)$ relative to y.[(1)]

We now consider any extension yz of y and compare $Q_{\text{LR}}(yz)$ with $Q_{\text{LR}}(y)$ and with $Q_{\text{LR}}(z)$. To facilitate the first of these two comparisons, we define a partial function $\alpha_{y,z} : Q_{\text{LR}}(y) \rightharpoonup Q$ as follows (Fig. 2b): for each $q \in Q_{\text{LR}}(y)$, we examine $\textsc{comp}_{q,|y|+1}(yz)$; if it hits right into some state r, we set $\alpha_{y,z}(q) := r$; if it hits left or loops, we leave $\alpha_{y,z}(q)$ undefined.

Fact 4a. *We have* $\alpha_{y,z}[Q_{\text{LR}}(y)] \supseteq Q_{\text{LR}}(yz)$ *and also* $Q_{\text{LR}}(yz) \subseteq Q_{\text{LR}}(z)$.[(2)]

Proof. Let $r \in Q_{\text{LR}}(yz)$. Then there exist p and u, v such that $c := \textsc{lcomp}_p(yz)$ appears in $\textsc{comp}(uyzv)$ and hits right into r (Fig. 2b). We know c crosses the y-z boundary at least once. Let q and q_* be the states right after the first crossing and after the last crossing, respectively. The prefix of c up to the first crossing is $c_1 := \textsc{lcomp}_p(y)$ and hits right into q, while the remaining suffix

is $c_2 := \textsc{comp}_{q,|y|+1}(yz)$ and hits right into r. The suffix of c after the last crossing is $c_* = \textsc{lcomp}_{q_*}(z)$ and hits right into r. Now, c_1 is a LR-traversal of y that appears in $\textsc{comp}(uyzv)$ and produces q, so $q \in Q_{\text{LR}}(y)$. By this and c_2, we know $\alpha_{y,z}(q) = r$. Therefore, $r \in \alpha_{y,z}[Q_{\text{LR}}(y)]$. Moreover, c_* is a LR-traversal of z that appears in $\textsc{comp}(uyzv)$ and produces r. Therefore, $r \in Q_{\text{LR}}(z)$. □

Symmetrically, we also consider the set $Q_{\text{RL}}(y)$ of RL-*outcomes* of y, consisting of all states that can be produced by RL-traversals of y inside full computations of M, and introduce the partial function $\beta_{z,y} : Q_{\text{RL}}(y) \rightharpoonup Q$ so that $\beta_{z,y}(q)$ is r if $\textsc{comp}_{q,|z|}(zy)$ hits left into r, or undefined if the computation loops or hits right. Then the following fact holds, which is symmetric to Fact 4a.

Fact 4b. *We have* $Q_{\text{RL}}(z) \supseteq Q_{\text{RL}}(zy)$ *and also* $Q_{\text{RL}}(zy) \subseteq \beta_{z,y}[Q_{\text{RL}}(y)]$.

By the first inclusion of Fact 4a, we know every distinct element of $Q_{\text{LR}}(yz)$ is hit via $\alpha_{y,z}$ by at least one distinct element of $Q_{\text{LR}}(y)$, so $|Q_{\text{LR}}(y)| \geq |Q_{\text{LR}}(yz)|$. Similarly, Fact 4b implies $|Q_{\text{RL}}(zy)| \leq |Q_{\text{RL}}(y)|$. Hence, extending a string in either direction can never increase the respective number of outcomes. Thus, sufficiently long extensions minimize this number. Such extensions are called *generic strings* and are defined as follows.

Definition. Let $L \subseteq \Sigma^*$. A string y is LR-*generic over* L if

$$y \in L \quad \& \quad \text{for all } yz \in L\text{: } |Q_{\text{LR}}(y)| = |Q_{\text{LR}}(yz)|\,.$$

Symmetrically, y is RL-*generic over* L if

$$y \in L \quad \& \quad \text{for all } zy \in L\text{: } |Q_{\text{RL}}(zy)| = |Q_{\text{RL}}(y)|\,.$$

If y is both LR-generic and RL-generic, then it is called *generic*.

Lemma 1. *Every* $\emptyset \neq L \subseteq \Sigma^*$ *admits both* LR- *and* RL-*generic strings.*(3) *And if* y_{L} *is* LR-*generic and* y_{R} *is* RL-*generic, then every* $y_{\text{L}}xy_{\text{R}} \in L$ *is generic over* L.

Proof. Suppose no LR-generic strings over L exist. Then every $y \in L$ has a $yz \in L$ such that $|Q_{\text{LR}}(y)| \neq |Q_{\text{LR}}(yz)|$, and thus $|Q_{\text{LR}}(y)| > |Q_{\text{LR}}(yz)|$. Hence, starting with any $y \in L$ and applying this rule ad infinitum, we find a sequence $y, yz_1, yz_1z_2, \ldots \in L$ in which $|Q_{\text{LR}}(\cdot)|$ keeps decreasing forever. This contradicts the obvious fact that $|Q_{\text{LR}}(\cdot)| \geq 0$. Hence, LR-generic strings over L exist. For RL-generic strings we work symmetrically.

For the final claim, it is enough to note that every right-extension $y_{\text{L}}z \in L$ of a LR-generic string y_{L} is also LR-generic (easily, by the definition). Similarly, every left-extension $zy_{\text{R}} \in L$ of a RL-generic string y_{R} is also RL-generic. □

Alternatively, genericity can be characterized via $\alpha_{y,z}$ and $\beta_{z,y}$, as follows.

Lemma 2. *Let* $y \in L \subseteq \Sigma^*$. *Then* y *is* LR-*generic over* L *iff* $\alpha_{y,z}$ *is total and bijective from* $Q_{\text{LR}}(y)$ *to* $Q_{\text{LR}}(yz)$ *for all* $yz \in L$.(4) *Similarly,* y *is* RL-*generic over* L *iff* $\beta_{z,y}$ *is total and bijective from* $Q_{\text{RL}}(y)$ *to* $Q_{\text{RL}}(zy)$ *for all* $zy \in L$.

Proof. We focus on the first equivalence (the second one follows symmetrically) and on the 'only if' direction —the 'if' direction is immediate, since the existence of any total bijection from $Q_{\text{LR}}(y)$ to $Q_{\text{LR}}(yz)$ implies $|Q_{\text{LR}}(y)| = |Q_{\text{LR}}(yz)|$.

Let y be LR-generic over L and pick $yz \in L$. We know $\alpha_{y,z}$ partially maps $Q_{\text{LR}}(y)$ to Q (by definition) and covers $Q_{\text{LR}}(yz)$ (Fact 4a). Namely, each $r \in Q_{\text{LR}}(yz)$ has a distinct $q \in Q_{\text{LR}}(y)$ with $\alpha_{y,z}(q) = r$. So, if there were $q \in Q_{\text{LR}}(y)$ with $\alpha_{y,z}(q)$ undefined or outside $Q_{\text{LR}}(yz)$ or equal to $\alpha_{y,z}(q')$ for another $q' \in Q_{\text{LR}}(y)$, we would have $|Q_{\text{LR}}(y)| > |Q_{\text{LR}}(yz)|$, contrary to y being generic. Hence, $\alpha_{y,z}(q)$ is defined and in $Q_{\text{LR}}(yz)$ and distinct, for all $q \in Q_{\text{LR}}(y)$. Namely, $\alpha_{y,z}$ is a total injection from $Q_{\text{LR}}(y)$ to $Q_{\text{LR}}(yz)$. By Fact 4a, it is also a surjection. □

4. Blocks

Let L be any non-empty property of strings, $\emptyset \neq L \subseteq \Sigma^*$. Pick any generic string ϑ over this property, and let $A := Q_{\text{LR}}(\vartheta)$ and $B := Q_{\text{RL}}(\vartheta)$ be the corresponding pair of sets of outcomes.

Every string of the form $\vartheta x \vartheta$ is called a *block* (*on* ϑ), and x is called the *infix* of it.[5] We say that the pair $(\alpha_x, \beta_x) := (\alpha_{\vartheta,x\vartheta}, \beta_{\vartheta x,\vartheta})$ are the *inner behavior* of M on the block. Note that $\alpha_x : A \rightharpoonup Q$ and $\beta_x : B \rightharpoonup Q$.

4.1. Pumping symbols

Blocks are useful because, in certain situations, they enable us to 'pump' symbols into the input without the machine noticing. Specifically, if we manage to force α_x and β_x to be identity functions, then the prefix ϑx and the suffix $x\vartheta$ of the block become 'invisible' to M, in the sense that the machine cannot distinguish between ϑ and the entire block $\vartheta x \vartheta$ in any environment u, v. The following lemma explains.

Lemma 3. *Suppose* $(\alpha_x, \beta_x) = (\text{id}_A, \text{id}_B)$. *Pick any two strings* u *and* v, *and let* $c_0, c_1, \ldots$ *and* $d_0, d_1, \ldots$ *be the decompositions of* COMP$(u\vartheta v)$ *and* COMP$(u\vartheta x\vartheta v)$ *relative to* ϑ *and to* $\vartheta x \vartheta$, *respectively. Then every* c_i *parallels* d_i *if* i *is even, or resembles* d_i *if* i *is odd. Thus,* M *behaves* (*accepts, rejects, or loops*) *identically on* $u\vartheta v$ *and* $u\vartheta x\vartheta v$.

Proof. Note that even-indexed c_i and d_i compute on $\vdash u$ or $v \dashv$, whereas odd-indexed c_i compute on ϑ, and odd-indexed d_i compute on $\vartheta x \vartheta$. It suffices to prove that *every* c_i *and* d_i *resemble each other and, if* i *is even, compute on the same string.* (For the even i, note that two resembling segments computing on the same string are necessarily parallel.)

We work inductively. If $i = 0$, then c_0 and d_0 are both L-computations on $\vdash u$ from q_s; so they are identical, and thus resembling, on the same string. For the inductive step, pick any $i > 0$ and assume the claim holds for c_{i-1} and d_{i-1}.

If i is even, then c_{i-1} computes on ϑ and d_{i-1} computes on $\vartheta x \vartheta$. We take cases on their common type. *If they are loops*, then no c_i and d_i exist, and we are done. *If they are* L*-turns or* RL*-traversals* (resp., R*-turns or* LR*-traversals*), then they both hit left (resp., right) into the same state p, causing c_i and d_i to

be R-computations (resp., L-computations) from p on $\vdash u$ (resp., $v\dashv$), and thus identical (resp., parallel), hence resembling, computations on the same string.

If i is odd, then c_{i-1} and d_{i-1} compute both on $\vdash u$ or both on $v\dashv$. We take cases on their common type and input. *If they are loops*, then no c_i and d_i exist, and we are done. *If they are* LR*-traversals on* $v\dashv$, then again no c_i and d_i exist. *If they are* LR*-traversals or* R*-turns on* $\vdash u$, then they both hit right into the same state p, causing c_i and d_i to be L-computations from p on ϑ and $\vartheta x\vartheta$, respectively. Since ϑ is a prefix of $\vartheta x\vartheta$, we know c_i is a prefix of d_i. Therefore, if c_i hits left or loops, then d_i remains identical to c_i, and thus resembles it. If instead c_i hits right, into some state q, then d_i crosses the ϑ-$x\vartheta$ boundary into q and continues with the suffix $\textsc{comp}_{q,|\vartheta|+1}(\vartheta x\vartheta)$, which we know hits right into q (because $\alpha_x(q) = \mathrm{id}_A(q) = q$); thus d_i again resembles c_i. *Each of the remaining cases* is either impossible or symmetric.

We now continue with the final claim. *If* M *halts on* $u\vartheta v$, then there exists a last c_m, with m even, which falls off $\dashv$. Hence, c_m is a LR-traversal of $v\dashv$, ending in q_a or q_r. Since c_m resembles d_m, we know d_m is also a LR-traversal of $v\dashv$, with the same last state. Hence, M halts on $u\vartheta x\vartheta v$ too, and decides identically. *If* M *loops on* $u\vartheta v$, then there are either infinitely many c_i, and thus also infinitely many resembling d_i, or a looping last c_m, with the resembling d_m also looping and last. Either way, M loops on $u\vartheta x\vartheta v$, too. □

Now, how can we force the inner behavior of M on a block into the identities? This we know how to do only when the generic string ϑ appears (not only at the two ends of the block, but also) multiple times inside the infix. That is, we can force identities on blocks of the form $\vartheta(x_1\vartheta x_2\vartheta\cdots\vartheta x_k)\vartheta$. The next few facts study how M computes on such blocks.

We start with the simple case of only one copy of ϑ inside the infix, namely blocks of the form $\vartheta(x\vartheta y)\vartheta$. The inner behavior of M on such blocks depends on its inner behavior on the sub-blocks $\vartheta x\vartheta$ and $\vartheta y\vartheta$, in the following manner.

Fact 5. *Let* $z = x\vartheta y$. *Then* $\alpha_x \circ \alpha_y \leq \alpha_z$ *and* $\beta_y \circ \beta_x \leq \beta_z$.[6] *In addition, if* α_z *is total and injective, then so is* α_x*; if* β_z *is total and injective, then so is* β_y.

Proof. For $\alpha_x \circ \alpha_y \leq \alpha_z$, let $p \in A$ and assume $(\alpha_x \circ \alpha_y)(p)$ is defined and equal to some $r \in Q$. Then $\alpha_x(p)$ is defined and equal to some $q \in Q$, and $\alpha_y(q)$ is defined and equal to r. By $\alpha_x(p) = q$, we know $c_x := \textsc{comp}_{p,|\vartheta|+1}(\vartheta x\vartheta)$ hits right into q (Fig. 3). By $\alpha_y(q) = r$, we also know $c_y := \textsc{comp}_{q,|\vartheta|+1}(\vartheta y\vartheta)$ hits right into r. Now, by concatenating c_x and c_y we get exactly $c_z := \textsc{comp}_{p,|\vartheta|+1}(\vartheta x\vartheta y\vartheta)$. Hence c_z hits right into r. Therefore $\alpha_z(p)$ is defined and equal to $(\alpha_x \circ \alpha_y)(p)$.

Now suppose α_z is total and injective. *If* α_x *is not total*, then $\alpha_x(p)$ is undefined for some $p \in A$, namely $c_x := \textsc{comp}_{p,|\vartheta|+1}(\vartheta x\vartheta)$ hits left or loops. But c_x is a prefix of $c_z := \textsc{comp}_{p,|\vartheta|+1}(\vartheta x\vartheta y\vartheta)$, so c_z also hits left or loops. Hence $\alpha_z(p)$ is undefined, and α_z is not total—a contradiction. *If* α_x *is not injective*, then $\alpha_x(p) = \alpha_x(p')$ for two distinct $p, p' \in A$, namely $c_x := \textsc{comp}_{p,|\vartheta|+1}(\vartheta x\vartheta)$ and $c'_x := \textsc{comp}_{p',|\vartheta|+1}(\vartheta x\vartheta)$ hit right into the same state. But c_x and c'_x are respectively prefixes of $c_z := \textsc{comp}_{p,|\vartheta|+1}(\vartheta x\vartheta y\vartheta)$ and $c'_z := \textsc{comp}_{p',|\vartheta|+1}(\vartheta x\vartheta y\vartheta)$, so

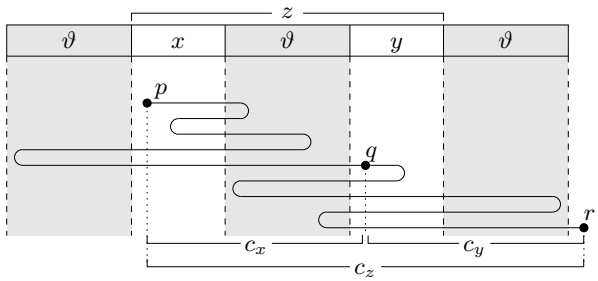

Figure 3: Computing on two overlapping blocks $\vartheta x \vartheta$ and $\vartheta y \vartheta$.

c_z and c'_z continue identically after the $\vartheta x \vartheta$-$y \vartheta$ boundary, hitting right into the same state. Hence $\alpha_z(p) = \alpha_z(p')$, and α_z is not injective—a contradiction.

The statements for the β's are proved by symmetric arguments. □

We now proceed to blocks of the form $\vartheta(x\vartheta x\vartheta \cdots x\vartheta x)\vartheta$, where the infix is multiple ϑ-separated copies of some x. We denote such infixes by

$$x^{(k)} := x(\vartheta x)^{k-1}$$

for $k \geq 1$. So, we now discuss blocks of the form $\vartheta x^{(k)}\vartheta = \vartheta(x\vartheta)^k = (\vartheta x)^k\vartheta$. Note that the usual law of exponents $(x^{(k)})^{(l)} = x^{(lk)}$ is valid for all k and l:

$$(x^{(k)})^{(l)} = x^{(k)}(\vartheta x^{(k)})^{l-1} = x(\vartheta x)^{k-1}(\vartheta x(\vartheta x)^{k-1})^{l-1} = x(\vartheta x)^{lk-1} = x^{(lk)}\,.$$

Extending Fact 5, the next fact shows the relationship between the inner behavior on the full block $\vartheta x^{(k)}\vartheta$ and the inner behavior on the basic block $\vartheta x \vartheta$.

Fact 6. *Let $k \geq 1$. Then $(\alpha_x)^k \leq \alpha_{x^{(k)}}$ and $(\beta_x)^k \leq \beta_{x^{(k)}}$. In addition, if $\alpha_{x^{(k)}}$ is total and injective, then so is α_x; if $\beta_{x^{(k)}}$ is total and injective, then so is β_x.*

Proof. Once again, we prove only the two claims for the α's.

For $(\alpha_x)^k \leq \alpha_{x^{(k)}}$, we use induction on k. Case $k = 1$ is trivial. For $k \geq 1$, assume $(\alpha_x)^k \leq \alpha_{x^{(k)}}$. Then $(\alpha_x)^{k+1} = \alpha_x \circ (\alpha_x)^k \leq \alpha_x \circ \alpha_{x^{(k)}}$ (Fact 1) and $\alpha_x \circ \alpha_{x^{(k)}} \leq \alpha_{x^{(k+1)}}$ (Fact 5 for $z = x\vartheta(x^{(k)}) = x\vartheta x(\vartheta x)^{k-1} = x(\vartheta x)^k = x^{(k+1)}$). So, $(\alpha_x)^{k+1} \leq \alpha_{x^{(k+1)}}$ (by transitivity of $\leq$), and we are done.

The second claim follows from Fact 5 when $z = x\vartheta(x^{(k-1)}) = x^{(k)}$. □

Finally, we are ready to state a sufficient condition for forcing the inner behavior of M into the setting of Lemma 3: if any infix of the form $x^{(k)}$ forces the inner behavior into just *permutations*, then infinitely many of the longer infixes of the same form force the inner behavior into just *identities*. The next fact says exactly this. The phrase $(\alpha_{x^{(k)}}, \beta_{x^{(k)}})$ *permute* (A, B) is shortcut for the condition that $\alpha_{x^{(k)}}$ is a permutation of A and $\beta_{x^{(k)}}$ is a permutation of B.

Fact 7. *If $(\alpha_{x^{(k)}}, \beta_{x^{(k)}})$ permute (A, B), then $(\alpha_{x^{(tlk)}}, \beta_{x^{(tlk)}}) = (\mathrm{id}_A, \mathrm{id}_B)$ for some $l \geq 1$ and all $t \geq 1$.*

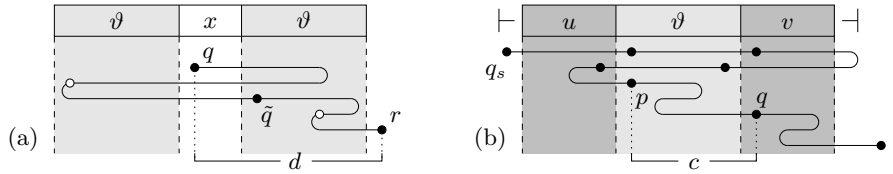

Figure 4: (a) A computation d that witnesses $\alpha_x(q) = r$, with two forward reversals. (b) An environment u, v and a computation c that witness $q \in A$.

Proof. Let $z := x^{(k)}$ and suppose that α_z and β_z are permutations of A and B.

Pick $l \geq 1$ so that each of these permutations becomes the corresponding identity after l iterations: $(\alpha_z)^l = \mathrm{id}_A$ and $(\beta_z)^l = \mathrm{id}_B$. Then $(\alpha_z)^l \leq \alpha_{z^{(l)}}$ (by Fact 6), where $z^{(l)} = (x^{(k)})^{(l)} = x^{(lk)}$. In other words, $\mathrm{id}_A \leq \alpha_{x^{(lk)}}$. Therefore $\alpha_{x^{(lk)}} = \mathrm{id}_A$ (since id_A is total). Similarly, $\beta_{x^{(lk)}} = \mathrm{id}_B$.

Now, consider any $t \geq 1$. By Fact 6, we know $(\alpha_{x^{(lk)}})^t \leq \alpha_{(x^{(lk)})^{(t)}}$. But $(\alpha_{x^{(lk)}})^t = (\mathrm{id}_A)^t = \mathrm{id}_A$ and $(x^{(lk)})^{(t)} = x^{(tlk)}$. Hence, we know $\mathrm{id}_A \leq \alpha_{x^{(tlk)}}$. Therefore $\alpha_{x^{(tlk)}} = \mathrm{id}_A$ (since id_A is total). Similarly, $\beta_{x^{(tlk)}} = \mathrm{id}_B$. □

4.2. Pumping reversals

Whenever we 'pump' symbols into a block 'under M's radar', some of the reversals that M performs on the block are unavoidably 'pumped' as well, into the computation of M on the longer input generated by the pumping. This effectively kills every hope of M maintaining a sublinear number of reversals.

The next lemma analyses this phenomenon. The phrase (A, B) *use reversals on* x is shortcut for the condition that some $\textsc{comp}_{p,|\vartheta|+1}(\vartheta x \vartheta)$ for $p \in A$ or some $\textsc{comp}_{p,|\vartheta x|}(\vartheta x \vartheta)$ for $p \in B$ contains at least one reversal.

Lemma 4. *If (A, B) use reversals on x and (α_x, β_x) permute (A, B), then it cannot be $r(n) = o(n)$.*

Proof. Since (A, B) use reversals on x, we know that there exists a computation $d := \textsc{comp}_{q,|\vartheta|+1}(\vartheta x \vartheta)$ with $q \in A$ (or $d := \textsc{comp}_{q,|\vartheta x|}(\vartheta x \vartheta)$ with $q \in B$, in which case we work symmetrically) that contains one or more reversals (Fig. 4a). In fact, d contains at least one *forward* reversal: since α_x permutes A, we know $\alpha_x(q)$ is defined, therefore d hits right, hence at least one of its reversals must be a forward one.

Since (α_x, β_x) permute (A, B), we also know (by Fact 7 for $k = 1$) that there exists $l \geq 1$ such that $(\alpha_{x^{(tl)}}, \beta_{x^{(tl)}}) = (\mathrm{id}_A, \mathrm{id}_B)$ for all $t \geq 1$.

Let $z := x^{(l)}$. Then $z^{(t)} = x^{(tl)}$ and thus $(\alpha_{z^{(t)}}, \beta_{z^{(t)}}) = (\mathrm{id}_A, \mathrm{id}_B)$, for all t. Using this, we show that each $d_t := \textsc{comp}_{q,|\vartheta|+1}(\vartheta z^{(t)} \vartheta)$ reverses a lot.

Claim. *For every $t \geq 1$, the computation d_t contains $\geq t$ forward reversals.*

Proof. By induction on t (Fig. 5). For $t = 1$, we have $d_1 = \textsc{comp}_{q,|\vartheta|+1}(\vartheta z^{(1)} \vartheta)$. Since $z^{(1)} = z = x^{(l)}$ and $l \geq 1$, we know the block $\vartheta z^{(1)} \vartheta$ has $\vartheta x \vartheta$ as prefix, causing d_1 to have d as prefix, and thus contain ≥ 1 forward reversals.

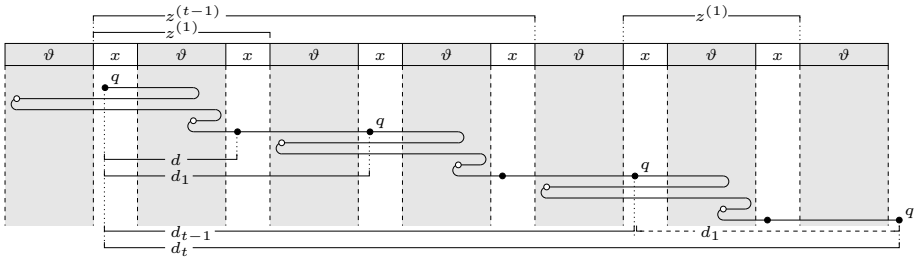

Figure 5: Pumping the 1 forward reversal of d into the t forward reversals of d_t, in the proof of the first Claim of Lemma 4; here $l = 2$ and $t = 3$.

For $t > 0$, we have $d_t = \textsc{comp}_{q,|\vartheta|+1}(\vartheta z^{(t)}\vartheta)$. Since $\vartheta z^{(t)}\vartheta = \vartheta z^{(t-1)}\vartheta z\vartheta$, the prefix of d_t up to the boundary $\vartheta z^{(t-1)}\vartheta$-$z\vartheta$ is d_{t-1}, and so the state after crossing this boundary is $\alpha_{z^{(t-1)}}(q) = \mathrm{id}_A(q) = q$. Thus, the remaining suffix $\textsc{comp}_{q,|\vartheta z^{(t-1)}\vartheta|+1}(\vartheta z^{(t-1)}\vartheta z\vartheta)$ parallels d_1. Hence, d_t contains the $\geq t-1$ forward reversals of d_{t-1} plus the ≥ 1 forward reversals of d_1, for a total of $\geq t$. ⊡

Note that, since the d_t are arbitrary computations, their existence does not prove that M performs many reversals. To establish this, we need to find many reversals inside *full* computations of M. This is our next step.

Since $q \in A = Q_{\mathrm{LR}}(\vartheta)$, there exist a state p and an environment u, v such that $c := \textsc{lcomp}_p(\vartheta)$ appears in $\hat{c} := \textsc{comp}(u\vartheta v)$ and hits right into q (Fig. 4b). Consider the family of inputs $w_t := u\vartheta z^{(t)}\vartheta v$ for $t \geq 1$, and the respective computations $\hat{c}_t := \textsc{comp}(w_t)$. We show that each $\hat{c}_t$ reverses a lot.

Claim. *For every $t \geq 1$, the computation $\hat{c}_t$ contains $\geq t$ forward reversals.*

Proof. By Lemma 3 and $(\alpha_{z^{(t)}}, \beta_{z^{(t)}}) = (\mathrm{id}_A, \mathrm{id}_B)$, we know c resembles a segment c_t in the decomposition of $\hat{c}_t$ relative to $\vartheta z^{(t)}\vartheta$ (Fig. 6). So, c_t is a L-computation on $\vartheta z^{(t)}\vartheta$ from p. Since ϑ is a prefix of $\vartheta z^{(t)}\vartheta$, the prefix of c_t up to the boundary ϑ-$z^{(t)}\vartheta$ is c, the state after crossing the boundary is q, and the suffix $\textsc{comp}_{q,|\vartheta|+1}(\vartheta z^{(t)}\vartheta)$ from then on parallels d_t. So, $\hat{c}_t$ also contains $\geq t$ forward reversals. ⊡

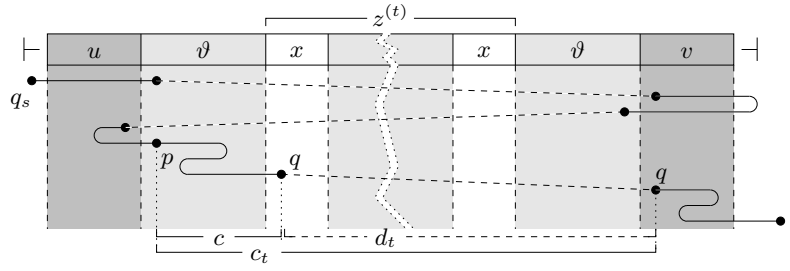

Figure 6: Placing d_t and its t forward reversals inside a segment c_t of a full computation $\hat{c}_t$, in the proof of the second Claim of Lemma 4.

We are almost done. We just observe that each $\hat{c}_t$ works on input length

$$n_t := |w_t| = |u\vartheta x^{(tl)}\vartheta v| = |u\vartheta(x\vartheta)^{tl}v| = l|x\vartheta| \cdot t + |u\vartheta v|\,.$$

Hence, the maximum number of reversals performed by M on n_t-long inputs is

$$r(n_t) \geq r(\hat{c}_t) \geq t = (1/l|x\vartheta|) \cdot n_t - |u\vartheta v|/l|x\vartheta|\,.$$

So, $r(n)$ exceeds a linear function inifinitely often. Hence, $r(n) \neq o(n)$. □

4.3. Block criterion

In Fact 6 we saw that the inner behavior of M on the short block $\vartheta x\vartheta$ affects its inner behavior on all longer blocks $\vartheta x^{(k)}\vartheta$ for $k \geq 1$. Not surprisingly, therefore, if we know how M decides on the long blocks, we can draw conclusions about its behavior on the short one. In a sense, this converts *global information about decisions* on a family of long inputs into *local information about computations* on a single short input.

Typically, the global information is implied by the assumption that M solves a certain problem $\mathfrak{L} = (L, \tilde{L})$, and thus accepts every long block which is in L but no long block which is in $\tilde{L}$. On the other hand, the local information takes the form of a criterion on α_x and β_x. In this section we argue our way through the conversion of the starting global assumption into the final local criterion.

So, let us call an infix x *positive*, *negative*, or *neutral* relative to a problem $\mathfrak{L} = (L, \tilde{L})$ if respectively $\vartheta x\vartheta$ is in L, in $\tilde{L}$, or in neither. In Section 5, we will encounter cases which satisfy the promise that

$$(\exists k \geq 1)\big(x^{(k)} \text{ is positive}\big) \quad \vee \quad (\forall k \geq 1)\big(x^{(k)} \text{ is negative}\big)$$

Whenever this holds, we will say that ϑ *and* x *respect* $\mathfrak{L}$; and that they *select* L or $\tilde{L}$, depending on whether the promise is met on its first half (namely, some $x^{(k)}$ is positive) or on its second half (namely, all $x^{(k)}$ are negative).

Now, if we also know that M solves $\mathfrak{L}$, then we can tell which of the two halves of the above promise is met using the local criterion whether (α_x, β_x) permute (A, B). The next fact assembles this criterion; the next lemma states the same criterion in a form which is easier to use.

Fact 8. *If positive $x^{(k)}$ exist, then (α_x, β_x) permute (A, B). In contrast, if all $x^{(k)}$ are negative and M solves $\mathfrak{L}$, then (α_x, β_x) do not permute (A, B).*

Proof. For the second implication, suppose M solves $\mathfrak{L}$ and (going for the contrapositive) assume that $(\alpha_x, \beta_x) = (\alpha_{x^{(1)}}, \beta_{x^{(1)}})$ permute (A, B) in order to find a non-negative $x^{(k)}$. Indeed, pick any $t \geq 1$ such that $(\alpha_{x^{(t\cdot 1)}}, \beta_{x^{(t\cdot 1)}}) = (\mathrm{id}_A, \mathrm{id}_B)$, among the infinitely many guaranteed by Fact 7, and let $k = t \cdot 1$. Then M behaves identically on ϑ and $\vartheta x^{(k)}\vartheta$ (by Lemma 3 with empty u, v) and thus accepts $\vartheta x^{(k)}\vartheta$ (since it accepts $\vartheta \in L$). Therefore, $\vartheta x^{(k)}\vartheta \notin \tilde{L}$, which implies that $x^{(k)}$ is not negative.

For the first implication, suppose $z := x^{(k)}$ is positive for some $k \geq 1$. We will prove that $\alpha_x : A \rightharpoonup Q$ is a permutation of A (omitting the symmetric proof that β_x is a permutation of B). To this end, we first need the following.

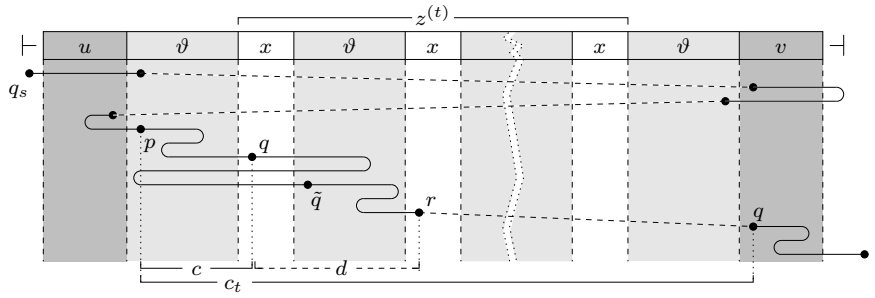

Figure 7: Proving that α_x keeps all its values in A, in the argument for Fact 8.

Claim. *(α_z, β_z) permute (A, B).*

Proof. Since z is positive, namely $\vartheta z \vartheta = \vartheta(x\vartheta)^k \in L$, we know $\alpha_z = \alpha_{\vartheta,(x\vartheta)^k}$ is a total bijection from $A = Q_{\text{LR}}(\vartheta)$ to $A' := Q_{\text{LR}}(\vartheta z \vartheta)$ (by Lemma 2). But $A' \subseteq A$ (by Fact 4a, since $\vartheta z \vartheta$ ends in ϑ) and $|A'| = |A|$ (since α_z is bijective), therefore $A' = A$. Thus, $\alpha_z = \alpha_{x^{(k)}}$ is a permutation of A. By a symmetric argument, $\beta_z = \beta_{x^{(k)}}$ is a permutation of B. ⊡

Now, in order to show that α_x permutes A, it is enough to prove two facts: first, that α_x is total and injective; and second, that $\alpha_x[A] \subseteq A$. The first fact follows directly from Fact 6 and the previous Claim, which implies that $\alpha_{x^{(k)}}$ is total and injective. For the second fact, we work as follows.

Let $r \in \alpha_x[A]$. Then there exists a state $q \in A = Q_{\text{LR}}(\vartheta)$ with $\alpha_x(q) = r$. In other words, there exist two states p, q and an environment u, v such that the computation $c := \text{LCOMP}_p(\vartheta)$ appears in $\hat{c} := \text{COMP}(u\vartheta v)$ and hits right into q (Fig. 4b), and $d := \text{COMP}_{q,|\vartheta|+1}(\vartheta x \vartheta)$ hits right into r (Fig. 4a). Note that c is an odd-indexed segment in the decomposition of $\hat{c}$ relative to ϑ. Now pick any $t \geq 1$ with $(\alpha_{z^{(t)}}, \beta_{z^{(t)}}) = (\alpha_{x^{(tk)}}, \beta_{x^{(tk)}}) = (\text{id}_A, \text{id}_B)$ (Fact 7). Lemma 3 says c resembles an odd-indexed segment c_t in the decomposition of $\hat{c}_t := \text{COMP}(u\vartheta z^{(t)} \vartheta v)$ relative to $\vartheta z^{(t)} \vartheta$ (Fig. 7). So, c_t is also a L-computation from p, on $\vartheta z^{(t)} \vartheta$. Since $\vartheta x \vartheta$ is a prefix of $\vartheta z^{(t)} \vartheta$, the prefix of c_t up to the first crossing of the right boundary of $\vartheta x \vartheta$ is c followed by a parallel of d. In particular, if $\tilde{q}$ is the state in d after the last crossing of the ϑx-ϑ boundary, then $\tilde{d} := \text{LCOMP}_{\tilde{q}}(\vartheta)$ hits right into r and appears in $\hat{c}_t = \text{COMP}((u\vartheta x)\vartheta(x^{(tk-1)}\vartheta v))$. Hence, $r \in Q_{\text{LR}}(\vartheta) = A$. □

Lemma 5. *Suppose that M solves $\mathfrak{L} = (L, \tilde{L})$, and that ϑ and x respect $\mathfrak{L}$. Then ϑ and x select L iff each outcome of ϑ is hit exactly once by the respective half of the inner behavior:*

$$\big(\forall r \in A\big)\big(|\alpha_x^{-1}(r)| = 1\big) \quad \& \quad \big(\forall r \in B\big)\big(|\beta_x^{-1}(r)| = 1\big)\,. \tag{2}$$

Proof. If ϑ and x select L, then there exist positive $x^{(k)}$, hence α_x permutes A (by Fact 8) and thus hits every $r \in A$ exactly once; similarly for β_x and B.

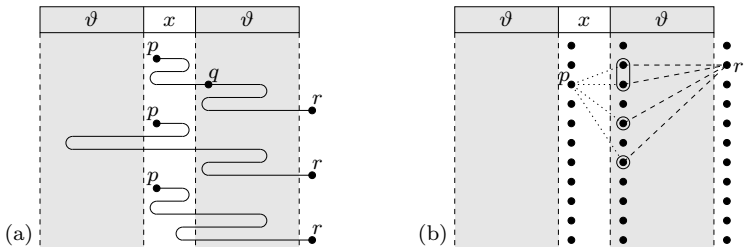

Figure 8: (a) Three right-hitting $\textsc{comp}_{p,|\vartheta|+1}(\vartheta x \vartheta)$: one simple (top), two non-simple. (b) Understanding S_p: each column is a copy of Q; circles and dashed edges represent the q for which $\textsc{lcomp}_q(\vartheta)$ hits right into r; dotted edges represent the $\delta_{\text{LR}}(p, x, q)$ that are being summed.

Conversely, if $\alpha_x : A \rightharpoonup Q$ hits every $r \in A$, then it is total and injective and keeps all its values inside A (or else its values would not be enough to cover A), hence it bijects A into A, i.e., it permutes A; similarly for β_x and B. Therefore, not all $x^{(k)}$ are negative (by Fact 8). So ϑ and x do not select $\tilde{L}$, but L. □

4.4. Block criterion under few reversals

In the special case where M uses sublinearly many reversals, criterion (2) in Lemma 5 can be simplified by replacing $\alpha_x^{-1}(r)$ and $\beta_x^{-1}(r)$ by two simpler sets, $\alpha_x^*(r)$ and $\beta_x^*(r)$, which we now introduce.

First, recall that $\alpha_x^{-1}(r)$ consists of all states $p \in A$ for which the computation $\textsc{comp}_{p,|\vartheta|+1}(\vartheta x \vartheta)$ hits right into r. Of course, every such computation is free to reach r after arbitrary meanders inside $\vartheta x \vartheta$. Suppose, however, that we restrict our attention only to computations which stay inside $x\vartheta$ and cross the x-ϑ boundary only once—we call these computations *simple* (Fig. 8a). Then, $\alpha_x^*(r)$ is the set of p which still manage to reach r:

$$\alpha_x^*(r) = \alpha^*_{\vartheta,x\vartheta}(r) := \{p \in A \mid (\exists q \in Q)(\textsc{lcomp}_p(x) \text{ hits right into } q \\ \& \ \textsc{lcomp}_q(\vartheta) \text{ hits right into } r)\}\,. \quad (3)$$

Symmetrically, we let $\beta_x^*(r) = \beta^*_{\vartheta x,\vartheta}(r)$ be the set of all states $p \in B$ for which $\textsc{comp}_{p,|\vartheta x|}(\vartheta x \vartheta)$ hits left into r having crossed the ϑx-ϑ and ϑ-$x\vartheta$ boundaries 0 and 1 times respectively.

A simple but important property of the new sets (one that the old sets do not share) is explained in the next fact, which uses the two boolean-valued functions $\delta_{\text{LR}}(\cdot\,,\cdot\,,\cdot\,)$ and $\delta_{\text{RL}}(\cdot\,,\cdot\,,\cdot\,)$ given by

$$\begin{aligned} \delta_{\text{LR}}(p, x, q) = 1 \quad &\Longleftrightarrow \quad \textsc{lcomp}_p(x) \text{ hits right into } q \\ \delta_{\text{RL}}(q, x, p) = 1 \quad &\Longleftrightarrow \quad \textsc{rcomp}_p(x) \text{ hits left into } q\,. \end{aligned}$$

The fact says that the sizes of $\alpha_x^*(r)$ and $\beta_x^*(r)$ can be expressed as simple sums of appropriate selections of the bits $\delta_{\text{LR}}(\cdot\,, x, \cdot\,)$ and $\delta_{\text{RL}}(\cdot\,, x, \cdot\,)$, respectively.

Fact 9. *For all* $r \in Q$: $\alpha_x^*(r) \subseteq \alpha_x^{-1}(r)$ *and* $\beta_x^*(r) \subseteq \beta_x^{-1}(r)$. *Moreover*:

$$|\alpha_x^*(r)| = \sum_{\substack{p \in A \ \&\ \textsc{lcomp}_q(\vartheta) \\ \text{hits right into } r}} \delta_{\mathrm{LR}}(p,x,q) \qquad |\beta_x^*(r)| = \sum_{\substack{p \in B \ \&\ \textsc{rcomp}_q(\vartheta) \\ \text{hits left into } r}} \delta_{\mathrm{RL}}(q,x,p)\,. \tag{4}$$

Proof. The inclusions are obvious. For the equality on the left, fix any $p \in A$ and consider the inner sum

$$S_p := \sum_{\substack{\textsc{lcomp}_q(\vartheta) \\ \text{hits right into } r}} \delta_{\mathrm{LR}}(p,x,q)\,.$$

This iterates over all q whose $\textsc{lcomp}_q(\vartheta)$ hits right into r, and counts how many of them are hit-right into by $\textsc{lcomp}_p(x)$ (Fig. 8b). By (3), every q counted this way is a witness for verifying that $p \in \alpha_x^*(r)$. Hence, S_p equals the number of such witnesses. Since M is deterministic, this number is ≤ 1 and thus

$$S_p = \begin{cases} 0 & \text{if } p \notin \alpha_x^*(r)\,, \\ 1 & \text{if } p \in \alpha_x^*(r)\,. \end{cases}$$

Consequently, the size of $\alpha_x^*(r)$ can be calculated by summing the S_p's,

$$|\alpha_x^*(r)| \;=\; \sum_{p \in A} S_p \;=\; \sum_{\substack{p \in A \ \&\ \textsc{lcomp}_q(\vartheta) \\ \text{hits right into } r}} \delta_{\mathrm{LR}}(p,x,q)\,,$$

and the equality is proven. The equality for $\beta_x^*(r)$ is proved symmetrically. □

We are now ready to prove the simplification of (2) that we promised.

Lemma 6. *Suppose that* M *solves* $\mathfrak{L} = (L, \tilde{L})$ *with* $r(n) = o(n)$ *reversals, and that* ϑ *and* x *respect* $\mathfrak{L}$. *Then* ϑ *and* x *select* L *iff each outcome of* ϑ *is hit by exactly one simple computation*:

$$\big(\forall r \in A\big)\big(|\alpha_x^*(r)| = 1\big) \quad \& \quad \big(\forall r \in B\big)\big(|\beta_x^*(r)| = 1\big)\,. \tag{5}$$

Proof. Suppose ϑ and x select L. Then (α_x, β_x) permute (A, B) (by Fact 8), therefore (A, B) do not use reversals (by Lemma 4, and since $r(n) = o(n)$).

Now pick any $r \in A$. We know $|\alpha_x^*(r)| \leq 1$, because $\alpha_x^*(r) \subseteq \alpha_x^{-1}(r)$ (by Fact 9) and $|\alpha_x^{-1}(r)| = 1$ (by Lemma 5). We also know $|\alpha_x^*(r)| \geq 1$, because the r-hitting $\textsc{comp}_{p,|\vartheta|+1}(\vartheta x \vartheta)$ for the unique $p \in \alpha_x^{-1}(r)$ uses no reversals (because (A, B) do not use reversals) and thus $p \in \alpha_x^*(r)$. Overall, $|\alpha_x^*(r)| = 1$. A similar argument applies for β_x^* and B.

Conversely, suppose criterion (5) holds. Then criterion (2) holds, too (since $\alpha_x^*(r) \subseteq \alpha_x^{-1}(r)$ and $\beta_x^*(r) \subseteq \beta_x^{-1}(r)$). So, ϑ and x select L (by Lemma 5). □

5. The hardness of liveness

We now proceed to the proof of the lower bound of Theorem 1. We pick an arbitrary $h \geq 1$ and suppose that the 2DFA M that we kept fixed throughout the previous sections is actually reading inputs over $\Sigma = \Sigma_h$ and solves OWL_h with $r(n) = o(n)$ reversals. We will show that the number of states in M must be exponential in the height h of the input alphabet, specifically $|Q| = \Omega(2^h)$.

To this end, we will first restrict our attention from the infinity of all possible instances of OWL_h down to a very specific family of inputs, which collectively capture the 'core' of the computational hardness overcome by M (Section 5.1). Next, we will use the fact that M decides correctly on all these 'hard instances' in order to argue our way through to the lower bound (Section 5.2).

5.1. The hard instances

We focus on the same family of instances that was used in [12, §3.2] for proving that SNFAs need exponentially many states against the complement of OWL_h. These instances are blocks of the form $\vartheta x^{(k)} \vartheta$, where $k \geq 1$ and ϑ and x are drawn from two families $(\vartheta_i)_{i \in \mathcal{I}}$ and $(x_i)_{i \in \mathcal{I}}$ of generic and single-symbol strings, respectively. Hence, describing the hard instances reduces to describing these two families of strings.

We start with the index set $\mathcal{I}$, which is all pairs of non-empty subsets of $[h]$,[1]

$$\mathcal{I} := \{\ (\alpha, \beta)\ \mid\ \emptyset \neq \alpha, \beta \subseteq [h]\ \}\,.$$

and is considered to be totally ordered by the rule

$$(\alpha', \beta') < (\alpha, \beta) \iff^{\text{def}} \langle\alpha'\rangle\langle\beta'\rangle <_{\text{b}} \langle\alpha\rangle\langle\beta\rangle\,,$$

where $\langle\cdot\rangle$ is the natural h-bit encoding for subsets of $[h]$, and $<_{\text{b}}$ stands for the natural ordering of $2h$-bit positive integers. E.g., for $h = 5$ and $\alpha = \{0, 1, 4\}$ we have $\langle\alpha\rangle = \texttt{10011}$; and if in addition $\beta = \{0, 2, 4\}$ and $\alpha' = \{0, 2\}$ and $\beta' = \{0, 2, 3\}$, then $(\alpha', \beta') < (\alpha, \beta)$ because the number $\langle\alpha'\rangle\langle\beta'\rangle = \texttt{00101 01101}$ is smaller than the number $\langle\alpha\rangle\langle\beta\rangle = \texttt{10011 10101}$.

Now, for each $(\alpha, \beta) \in \mathcal{I}$ consider the property of having connectivity $\alpha \times \beta$:

$$L_{\alpha,\beta} := \{\ z \in \Sigma^*\ \mid\ \xi(z) = \alpha \times \beta\ \}\,.$$

We can verify that generic strings over $L_{\alpha,\beta}$ exist, as follows. Pick any LR-generic string ϑ_{L} and any RL-generic string ϑ_{R} (guaranteed to exist by Lemma 1) and join them into $\vartheta := \vartheta_{\text{L}}\eta\vartheta_{\text{R}}$ with the 'reset' symbol $\eta := [h] \times [h]$ of all possible arrows. Then the connectivity of ϑ is (easily) also $\alpha \times \beta$. Hence $\vartheta \in L_{\alpha,\beta}$, which implies that ϑ is generic over $L_{\alpha,\beta}$ (by Lemma 1).

[1] Here α and β (without subscripts) denote subsets of $[h]$. These names preserve notational symmetry, and should cause no confusion with the names (with subscripts) of the two partial functions in the inner behavior of M.

We are now ready to define the strings ϑ_i and x_i, for each $i = (\alpha, \beta) \in \mathcal{I}$. First, ϑ_i is any of the (infinitely many) generic strings over $L_i = L_{\alpha,\beta}$. Second, $x_i := \overline{\beta \times \alpha}$ is the (unique) 1-long string consisting of all arrows not in $\beta \times \alpha$.

Consider all short blocks of the form $\vartheta_i x_j \vartheta_i$ that can be constructed from these strings. We naturally picture these blocks on a $|\mathcal{I}| \times |\mathcal{I}|$ matrix. Cell (i, j) of this matrix hosts the block $\vartheta_i x_j \vartheta_i$ along with copies of all objects that are associated with it in Lemma 6: the sets and functions

$$A_i := Q_{\text{LR}}(\vartheta_i) \quad B_i := Q_{\text{RL}}(\vartheta_i) \qquad \alpha^*_{i,j} := \alpha^*_{\vartheta_i, x_j \vartheta_i} \quad \beta^*_{i,j} := \beta^*_{\vartheta_i x_j, \vartheta_i}\,.$$

Crucially, the assumptions of Lemma 6 are satisfied in every single cell, whereas its conclusions follow a very simple pattern on and below the main diagonal (i.e., when $i \geq j$). The next fact proves this observation. Its statement uses

$$L_\emptyset := \{z \in \Sigma^* \mid \xi(z) = \emptyset\} = \overline{\text{OWL}_h}$$

as an extra name for the dead strings, for symmetry with the L_i.

Fact 10. *For all $i, j \in \mathcal{I}$, the assumptions of Lemma 6 are satisfied by ϑ_i, x_j, and $\mathfrak{L}_i = (L_i, L_\emptyset)$. Moreover, if $i > j$ then ϑ_i and x_j select L_i; in contrast, if $i = j$ then ϑ_i and x_j select $L_\emptyset$.*

Proof. Let $i = (\alpha, \beta)$ and $j = (\alpha', \beta')$. We start by checking the assumptions of Lemma 6. Clearly, M solves $\mathfrak{L}_i$ (since all strings in L_i are live and all strings in $L_\emptyset$ are dead) with $r(n) = o(n)$ (by assumption), and ϑ_i is generic over L_i (by selection). To show that ϑ_i and x_j respect $\mathfrak{L}_i$, we take cases.

If $\vartheta_i x_j \vartheta_i$ is dead, then so is every $\vartheta_i (x_j \vartheta_i)^k$ for $k \geq 1$ (since every extension of a dead string is also dead). Hence, all $(x_j)^{(k)}$ are negative.

If $\vartheta_i x_j \vartheta_i$ is live, then some path $a^* \rightsquigarrow b^*$ connects the two outer columns, for $a^*, b^* \in [h]$ (Fig. 9, left). If b', a' are the nodes visited by this path on the two columns of x_j, then the path is of the form $a^* \rightsquigarrow b' \to a' \rightsquigarrow b^*$ and ϑ_i contains the two paths $a^* \rightsquigarrow b'$ and $a' \rightsquigarrow b^*$. Hence $(a^*, b'), (a', b^*) \in \xi(\vartheta_i) = \alpha \times \beta$, which implies that $b' \in \beta$ and $a' \in \alpha$. Now pick any $a, b \in [h]$ and consider node a of the leftmost column and node b of the rightmost column of $\vartheta_i x_j \vartheta_i$. If $(a, b) \notin (\alpha, \beta)$, then $a \notin \alpha$ or $b \notin \beta$, hence at least one of the nodes cannot 'see through' ϑ_i, and thus the nodes do not connect in $\vartheta_i x_j \vartheta_i$. In contrast, if $(a, b) \in (\alpha, \beta)$, then $a \in \alpha$ and $b \in \beta$, hence $(a, b'), (a', b) \in \xi(\vartheta_i)$, and thus the nodes connect via a path of the form $a \rightsquigarrow b' \to a' \rightsquigarrow b$. Overall $\xi(\vartheta_i x_j \vartheta_i) = \alpha \times \beta$, namely $\vartheta_i x_j \vartheta_i \in L_i$. Hence, $(x_j)^{(1)}$ is positive.

Since the above two cases are exhaustive, we conclude that ϑ_i and x_j respect $\mathfrak{L}_i$. Moreover, we showed that *ϑ_i and x_j select L_i iff $\vartheta_i x_j \vartheta_i$ is live.*

For the second half of the statement, we examine each case separately.

If $i > j$, then the binary number $\langle \alpha \rangle \langle \beta \rangle$ is strictly greater than the binary number $\langle \alpha' \rangle \langle \beta' \rangle$. This implies that at least one of the 1's in $\langle \alpha \rangle \langle \beta \rangle$ corresponds to a 0 in $\langle \alpha' \rangle \langle \beta' \rangle$. Therefore, at least one of $\alpha \not\subseteq \alpha'$ or $\beta \not\subseteq \beta'$ is true. Suppose $\beta \not\subseteq \beta'$ (if $\alpha \not\subseteq \alpha'$, apply a similar argument). Pick any $a^* \in \alpha$ and $b' \in \beta \setminus \beta'$ and $a' \in \alpha$ and $b^* \in \beta$ (Fig. 9, left). Then $(a^*, b') \in \xi(\vartheta_i)$ and $(b', a') \in \xi(x_j)$ and

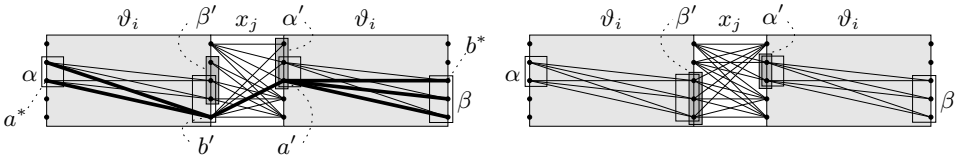

Figure 9: The connectivity of hard blocks below the diagonal (left) and on the diagonal (right).

$(a', b^*) \in \xi(\vartheta_i)$, therefore $\vartheta_i x_j \vartheta_i$ contains the path $a^* \rightsquigarrow b' \to a' \rightsquigarrow b^*$. Hence, it is live. By our previous conclusion, this implies that ϑ_i and x_j select L_i.

If $i = j$ then $\xi(\vartheta_i) = \alpha \times \beta$ and $\xi(x_j) = \overline{\beta \times \alpha}$ (Fig. 9, right). Then $\vartheta_i x_j \vartheta_i$ is dead, because a path of the form $a^* \rightsquigarrow b' \to a' \rightsquigarrow b^*$ needs to have $(a^*, b') \in \xi(\vartheta_i)$ and $(b', a') \in \xi(x_j)$ and $(a', b^*) \in \xi(\vartheta_i)$, where the membership in the middle says $(b', a') \notin \beta \times \alpha$ and the two others imply $b' \in \beta$ and $a' \in \alpha$, which is impossible. Hence, by our previous conclusion, ϑ_i and x_j select $L_\emptyset$. □

5.2. The lower bound

Now consider the following two collections of experiments.

In the first collection, we perform one experiment for every generic string ϑ_i and every state $r \in Q$. With ϑ_i and r fixed, we let x_j range over all possibilities and observe how the sizes of the sets $\alpha^*_{i,j}(r)$ and $\beta^*_{i,j}(r)$ vary. The result of our observation is a pair of $1 \times |\mathcal{I}|$ vectors,

$$\mathrm{a}_{i,r} := \big(|\alpha^*_{i,j}(r)| \big)_{j \in \mathcal{I}} \qquad \text{and} \qquad \mathrm{b}_{i,r} := \big(|\beta^*_{i,j}(r)| \big)_{j \in \mathcal{I}} .$$

Repeating for every possible ϑ_i and r, we arrive at the two sets of vectors

$$\mathcal{A} := \{ \mathrm{a}_{i,r} \mid i \in \mathcal{I}, r \in Q \} \qquad \text{and} \qquad \mathcal{B} := \{ \mathrm{b}_{i,r} \mid i \in \mathcal{I}, r \in Q \} .$$

In the second collection, we perform one experiment for every $p, q \in Q$. With p and q fixed, we again let x_j range over all possibilities. This time we observe the bits $\delta_{\mathrm{LR}}(p, x_j, q)$ and $\delta_{\mathrm{RL}}(q, x_j, p)$. The result is a pair of $1 \times |\mathcal{I}|$ vectors,

$$\mathrm{u}_{p,q} := \big(\delta_{\mathrm{LR}}(p, x_j, q) \big)_{j \in \mathcal{I}} \qquad \text{and} \qquad \mathrm{v}_{q,p} := \big(\delta_{\mathrm{RL}}(q, x_j, p) \big)_{j \in \mathcal{I}} .$$

Repeating the same for every p and q, we arrive at the sets of vectors

$$\mathcal{U} := \{ \mathrm{u}_{p,q} \mid p, q \in Q \} \qquad \text{and} \qquad \mathcal{V} := \{ \mathrm{v}_{q,p} \mid p, q \in Q \} .$$

The proof now concludes with two more facts. First, within $\mathcal{A} \cup \mathcal{B}$ we can find $\geq |\mathcal{I}| - 1$ vectors which are linearly independent (Fact 11a). Second, every vector in $\mathcal{A} \cup \mathcal{B}$ is a linear combination of the $\leq 2|Q|^2$ distinct vectors of $\mathcal{U} \cup \mathcal{V}$ (Fact 11b). Hence, the dimension of the span of the vectors of $\mathcal{U} \cup \mathcal{V}$ should be large enough to accomodate all linearly independent vectors of $\mathcal{A} \cup \mathcal{B}$ (see, e.g., [13, Prop. 14.1]). Namely,

$$2|Q|^2 \geq |I| - 1 .$$

Since $|\mathcal{I}| = (2^h - 1)^2$, it follows that $|Q| = \Omega(2^h)$ and the proof is complete.

Fact 11a. *The set $\mathcal{A} \cup \mathcal{B}$ contains $|\mathcal{I}| - 1$ linearly independent vectors.*

Proof. Within $\mathcal{A} \cup \mathcal{B}$ we will find a family of vectors $(\mathrm{c}_i)_{i \in \mathcal{I}}$ such that

$$i > j \implies \mathrm{c}_i(j) = 1 \qquad \text{and} \qquad i = j \implies \mathrm{c}_i(j) = 0\,, \tag{6}$$

for all $i, j \in \mathcal{I}$. This will be enough. Because then the numbers $\mathrm{c}_i(j)$ form a $|\mathcal{I}| \times |\mathcal{I}|$ matrix with 0s on the diagonal and 1s below it, which has rank $|\mathcal{I}| - 1$ (easily), and thus $|\mathcal{I}| - 1$ of the c_i must be linearly independent.

To select these vectors, we pick any $i \in \mathcal{I}$ and argue as follows. First of all, we know that ϑ_i and x_i select $L_\emptyset$ (Fact 10). Therefore, there exist outcomes r which are *not* hit by exactly 1 simple computation (Lemma 6):

$$r \in A_i \quad \& \quad |\alpha^*_{i,i}(r)| \neq 1 \qquad \text{or} \qquad r \in B_i \quad \& \quad |\beta^*_{i,i}(r)| \neq 1\,.$$

At least one of these r must, in fact, be hit by 0 simple computations—otherwise, every $r \in A_i$ is hit by ≥ 1 value of $\alpha_{i,i}$, every $r \in B_i$ is hit by ≥ 1 value of $\beta_{i,i}$, and at least one of all these r is hit by ≥ 2 values, for a total of $\geq |A_i| + |B_i| + 1$ values of $\alpha_{i,i}$ and $\beta_{i,i}$ together, a contradiction. Pick r_i to be any of these unhit outcomes. Then

$$r_i \in A_i \quad \& \quad |\alpha^*_{i,i}(r_i)| = 0 \qquad \text{or} \qquad r_i \in B_i \quad \& \quad |\beta^*_{i,i}(r_i)| = 0\,.$$

Depending on whether we select r_i from A_i or B_i, we respectively set

$$\mathrm{c}_i := \mathrm{a}_{i,r_i} \qquad \text{or} \qquad \mathrm{c}_i := \mathrm{b}_{i,r_i}$$

and the selection of our family of vectors is complete.

We now prove that our selection satisfies (6). Let $i, j \in \mathcal{I}$. Suppose $\mathrm{c}_i = \mathrm{a}_{i,r_i}$ (if $\mathrm{c}_i = \mathrm{b}_{i,r_i}$, we argue similarly). *For the first conjunct*, suppose $i > j$. Then ϑ_i and x_j select L_i (Fact 10). Thus, each $r \in A_i$ and $r \in B_i$ is hit by exactly 1 simple computation (Lemma 6). In particular, this is true of r_i. Hence,

$$\mathrm{c}_i(j) = \mathrm{a}_{i,r_i}(j) = |\alpha^*_{i,j}(r_i)| = 1\,.$$

For the second conjunct, suppose $i = j$. Then, by the selection of r_i directly,

$$\mathrm{c}_i(j) = \mathrm{c}_i(i) = \mathrm{a}_{i,r_i}(i) = |\alpha^*_{i,i}(r_i)| = 0\,.$$

Therefore, both conjuncts of (6) are true and the proof is complete. □

Fact 11b. *Every vector in $\mathcal{A} \cup \mathcal{B}$ is a linear combination of vectors from $\mathcal{U} \cup \mathcal{V}$.*

Proof. Let $i \in \mathcal{I}$ and $r \in Q$. For all $j \in \mathcal{I}$, the left equality of (4) implies that

$$\mathrm{a}_{i,r}(j) = |\alpha^*_{i,j}(r)| = |\alpha^*_{\vartheta_i, x_j \vartheta_i}(r)| = \sum_{\substack{p \in A_i \ \&\ \textsc{lcomp}_q(\vartheta_i) \\ \text{hits right into } r}} \delta_{\mathrm{LR}}(p, x_j, q) = \sum_{\substack{p \in A_i \ \&\ \textsc{lcomp}_q(\vartheta_i) \\ \text{hits right into } r}} \mathrm{u}_{p,q}(j)$$

whereas the right equality of (4) supports a similar chain of equations for $\mathrm{b}_{i,r}(j)$. Therefore, $\mathrm{a}_{i,r}$ and $\mathrm{b}_{i,r}$ are the following linear combinations

$$\mathrm{a}_{i,r} = \sum_{\substack{p \in A_i \ \&\ \textsc{lcomp}_q(\vartheta_i) \\ \text{hits right into } r}} \mathrm{u}_{p,q} \qquad \text{and} \qquad \mathrm{b}_{i,r} = \sum_{\substack{p \in B_i \ \&\ \textsc{rcomp}_q(\vartheta_i) \\ \text{hits left into } r}} \mathrm{v}_{q,p}$$

of vectors from $\mathcal{U}$ and of vectors from $\mathcal{V}$, respectively. □

6. Weaker than general

Let $h \geq 1$. In this section we describe a language that needs $O(h^2)$ states on a general 2DFA but $\Omega(2^h)$ states on every 2DFA with few reversals.

Our witness will be a restriction of OWL_h to instances of a special form. The restriction will be *strong enough* so that the problem becomes easy for general 2DFAs, but also *weak enough* so that the problem remains hard for 2DFAs with few reversals. Once we describe it, it will be straightforward to design a small general 2DFA solver, and equally straightforward to turn the argument of Section 5 into one that is valid even for the restricted language.

This strategy is not new. It was applied in [14] in order to convert the separation of small SNFAs from small SDFAs (by [3]) into a separation of small general 2DFAs from small SDFAs. It was also applied in [15, §3.6] in order to convert the separation of small co-nondeterministic SFAs from small SNFAs (by [12]; also in [15, §3]) into a separation of small general 2DFAs from small SNFAs. Our restriction will be exactly the one used in this second application.

6.1. The witness

To describe the special form of the restricted instances, we need the 'reset' symbol $\eta = [h] \times [h]$, from Section 5.1. We also need the restriction Σ'_h of Σ_h to the 2^h 'parallel' symbols of the form $\{(a, a) \mid a \in \alpha\}$ for $\alpha \subseteq [h]$; for example, the leftmost symbol in Fig. 1a is in Σ'_5, for $\alpha = \{1, 2, 3, 4\}$. Now, we restrict OWL_h to strings of the form (Fig. 10):

$$\text{SEGMENTS}_h := \Sigma'_h \big(\eta \Sigma'_h \Sigma_h \Sigma'_h\big)^* \eta \Sigma'_h \,.$$

Namely, every string must start and end with a parallel symbol and, in between, it must consist of one or more resets separated by 3-symbol *segments*, each of which consists of an unrestricted symbol between two parallel ones. Clearly, *every such string is live iff the first and last symbols are non-empty and all segments are live*. Intuitively, the copies of η 'segment' the task into 3-symbol pieces by resetting liveness every four symbols. We use the name

$$\text{SEGMENTED OWL}_h := \text{SEGMENTS}_h \cap \text{OWL}_h$$

for the problem of checking liveness on such strings.

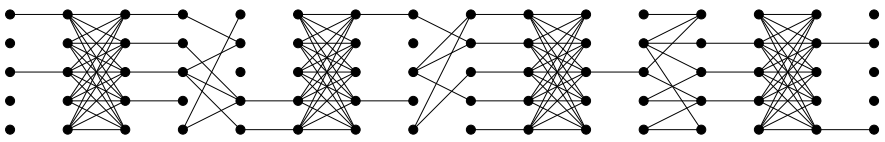

Figure 10: A string in SEGMENTS_5, where four resets delimit three segments.

6.2. The argument

Given the last observation, a small 2DFA algorithm for SEGMENTED OWL$_h$ is obvious: we just check that the input is of the correct form, that the first and last symbols are non-empty, and that every segment is live. For the first two checks, $O(1)$ states are enough. For the last check, we may iterate a $O(h^2)$-state depth-first search. Overall, a general 2DFA can solve SEGMENTED OWL$_h$ with $O(h^2)$ states. Not surprisingly, this algorithm performs $\Theta(n)$ reversals.

In contrast, 2DFAs with $o(n)$ reversals still need $\Omega(2^h)$ states to solve the problem. To prove this, we repeat the argument of Section 5, this time being careful to select all hard instances from within SEGMENTS$_h$.

Specifically, for each $i = (\alpha, \beta) \in \mathcal{I}$, we replace property L_i with the subset

$$L'_i := \text{SEGMENTS}_h \cap L_i$$

of strings which have (not only the correct connectivity $\alpha \times \beta$, but also) the correct form. Easily, *these are the strings in* SEGMENTS$_h$ *where all segments are live and the parallel first and last symbols are induced respectively by* α *and* β.

Much like L_i, the restricted property L'_i also admits generic strings. First, Lemma 1 guarantees two strings ϑ'_{L} and ϑ'_{R} which are LR-generic and RL-generic over L'_i. From them, we get $\vartheta' := \vartheta'_{\text{L}}\eta\vartheta'_{\text{R}}$. This is also in L'_i, because it inherits from ϑ'_{L} and ϑ'_{R} the correct first symbol, the correct last symbol, and a number of segments which are all live, plus a brand-new segment 'in the middle' (formed by the non-empty last symbol of ϑ'_{L}, the joining η, and the non-empty first symbol of ϑ'_{R}) which is clearly live as well. Thus ϑ' is generic over L'_i (by Lemma 1).

Finally, Fact 10 remains valid if we replace every L_i with L'_i and every ϑ_i with a ϑ'_i generic over L'_i. This follows from the observation that, because all blocks $\vartheta'_i x_j \vartheta'_i$ are in SEGMENTS$_h$, the validity of the argument depends only on the connectivities of the strings appearing in it, and these have not changed.

The rest of the proof remains the same.

7. Stronger than sweeping

Let $h \geq 1$. In this section we describe a language that needs $O(h)$ states on a 2DFA with 2 reversals but $2^{\Omega(h)}$ states on every SDFA.

For our lower bound, we use the "*hardness propagation*" framework of [16]. The main idea there was that, in some cases, if a problem $\mathfrak{L}$ needs 'many' states on automata of a certain type X, then by transforming it appropriately we can construct a *harder* problem $\mathfrak{L}'$ which needs 'many' states on automata of a *more powerful* type X'. Thus, hardness 'propagates' upwards from $\mathfrak{L}$ versus X to $\mathfrak{L}'$ versus X'. Several lemmata of this form were proven in [16]. Applying them in succession, one could easily transform, e.g., a problem which is hard for 1DFAs into a problem which is hard for SDFAs.

We do the same here. We start with a problem $\mathfrak{J}$ which is easily seen to require $\geq 2^h$ states on 1DFAs, and transform it into a problem $\mathfrak{J}'$ which requires $2^{\Omega(h)}$ states on SDFAs. Before this, we need to recall some of the problem transformations introduced in [16], along with one of the hardness propagation

lemmata. We then define a new transformation and prove a new hardness propagation lemma for it. Finally, we use the old and new transformations and lemmata to define our witness and establish the separation.

7.1. Hardness propagation

Let $\mathfrak{L} = (L, \tilde{L})$ be a problem. The *complement* and the *reverse* of $\mathfrak{L}$ are

$$\neg\mathfrak{L} = (\tilde{L}, L) \qquad \text{and} \qquad \mathfrak{L}^{\mathsf{R}} = (L^{\mathsf{R}}, \tilde{L}^{\mathsf{R}})\,.$$

For # any fresh symbol, the *conjunctive star* of $\mathfrak{L}$ is the problem of checking whether a #-delimited string of instances of $\mathfrak{L}$ contains only positive instances, whereas the *disjunctive star* asks whether a positive instance exists:

$$\begin{aligned}\textstyle\bigwedge\mathfrak{L} &:= \big(\ \{\#x_1\#\cdots\#x_l\# \mid (\forall i)(x_i \in L)\},\ \ \{\#x_1\#\cdots\#x_l\# \mid (\exists i)(x_i \in \tilde{L})\}\ \big)\\ \textstyle\bigvee\mathfrak{L} &:= \big(\ \{\#x_1\#\cdots\#x_l\# \mid (\exists i)(x_i \in L)\},\ \ \{\#x_1\#\cdots\#x_l\# \mid (\forall i)(x_i \in \tilde{L})\}\ \big)\,,\end{aligned}$$

where the form $\#x_1\#\cdots\#x_l\#$ assumes $l \geq 0$ and every $x_i \in L \cup \tilde{L}$. The equalities

$$\begin{aligned}\neg\textstyle\big(\bigwedge\mathfrak{L}\big) &= \bigvee\neg\mathfrak{L}\\ \neg\textstyle\big(\bigvee\mathfrak{L}\big) &= \bigwedge\neg\mathfrak{L}\end{aligned} \qquad \neg\big(\mathfrak{L}^{\mathsf{R}}\big) = \big(\neg\mathfrak{L}\big)^{\mathsf{R}} \qquad \begin{aligned}\textstyle\big(\bigwedge\mathfrak{L}\big)^{\mathsf{R}} &= \bigwedge\mathfrak{L}^{\mathsf{R}}\\ \textstyle\big(\bigvee\mathfrak{L}\big)^{\mathsf{R}} &= \bigvee\mathfrak{L}^{\mathsf{R}}\,.\end{aligned}$$

are easy to verify, directly from these definitions.

We will use the following hardness propagation lemma [16, Lemma 5]:

Lemma 7. *If no m-state* 1DFA *solves $\mathfrak{L}$, then no $\cap_{\mathrm{L}}$*1DFA *with m-state components solves $\bigvee\mathfrak{L}$.*

Here, '$\cap_{\mathrm{L}}$1DFA' stands for 'left-sided parallel intersection automaton'. In general, a *parallel intersection automaton* is a pair $M = (\mathcal{A}, \mathcal{B})$ of disjoint families of 1DFAs. To run M on an input z means to run each component 1DFA $D \in \mathcal{A} \cup \mathcal{B}$ on z separately and record the result, but with a twist: every $D \in \mathcal{A}$ reads z from left to right (as usual), whereas every $D \in \mathcal{B}$ reads z from right to left (i.e., it reads z^{R}). Moreover, each D may hang within z. We say that M *accepts* z if all components accept. If $\mathcal{A} = \emptyset$ or $\mathcal{B} = \emptyset$, then M is respectively *right-sided* (a $\cap_{\mathrm{R}}$1DFA) or *left-sided* (a $\cap_{\mathrm{L}}$1DFA). If the definition is modified so that M accepts if *some* component accepts, then M is a *parallel union automaton*, which again can be *right-sided* (a $\cup_{\mathrm{R}}$1DFA) or *left-sided* (a $\cup_{\mathrm{L}}$1DFA). We will need the following easy observations.

Fact 12. *If a $\cup_{\mathrm{R}}$*1DFA *with m-state components solves $\mathfrak{L}$, then a $\cup_{\mathrm{L}}$*1DFA *with m-state components solves $\mathfrak{L}^{\mathsf{R}}$. If a $\cup_{\mathrm{L}}$*1DFA *with m-state components solves $\mathfrak{L}$, then a $\cap_{\mathrm{L}}$*1DFA *with $(m+1)$-state components solves $\neg\mathfrak{L}$.*

Proof. For the first statement, just swap the two 1DFA families. For the second statement, just add a rejecting sink state to each component (to rule out rejections by hanging) and then complement the sets of accepting states. □

Finally, we also recall the following 'core' problem, defined over the alphabet $[h] \cup \mathbb{P}([h])$ of numbers and sets of numbers in $[h]$:

$$\mathfrak{J} = \text{SET NUM}_h := \big(\ \{\alpha i \mid \alpha \subseteq [h] \ \ \& \ \ i \in \alpha\}, \ \ \{\alpha i \mid \alpha \subseteq [h] \ \ \& \ \ i \in \overline{\alpha}\} \ \big), \quad (7)$$

where we are given a set α and a number i (in this order), and we must check that $i \in \alpha$. It is easy to prove that $\mathfrak{J}$ requires $\geq 2^h$ states on every 1DFA, whereas its reversal $\mathfrak{J}^{\text{R}}$ (where the number is given before the set) requires $\leq h$.

This concludes our summary of facts from [16]. We now add to these a new problem transformation, along with a new hardness propagation lemma for it.

7.2. Ordered star

Let $\mathfrak{L}_1 = (L_1, \tilde{L}_1)$ and $\mathfrak{L}_2 = (L_2, \tilde{L}_2)$ be two problems of disjoint promises. We define a new problem, where the input is promised to be a string $\#x_1\#\cdots\#x_l\#$ of #-delimited instances of $\mathfrak{L}_1$ and $\mathfrak{L}_2$. Of course, each x_i may be positive or negative in the respective problem. We are also promised that, although negative instances may mingle freely, positive instances do not: *one of the problems places all its positive instances before all positive instances of the other problem.* Note that this extra promise is vacuously true whenever one of the problems contributes no positive instances at all. Now, under these promises, our task depends on whether both problems contribute positive instances: if so, we must check that the one which places its positive instances first is $\mathfrak{L}_1$; if not, we must check that neither problem contributes any positive instance. In summary:

> Given a string $\#x_1\#\cdots\#x_l\#$ of #-delimited instances of $\mathfrak{L}_1$ and $\mathfrak{L}_2$ where all positive instances of one of the problems appear before all positive instances of the other, check that *either* both problems contribute positive instances and the one that places them first is $\mathfrak{L}_1$ *or* neither problem contributes any positive instance.

We call this problem the *ordered star* of $\mathfrak{L}_1$ and $\mathfrak{L}_2$ and denote it by $\mathfrak{L}_1 < \mathfrak{L}_2$.

The intuition behind this definition is that, under certain assumptions, $\mathfrak{L}_1 < \mathfrak{L}_2$ is easy for a 2DFA with just 2 reversals but hard for a SDFA. Specifically, suppose that checking $\mathfrak{L}_1$ is *hard* during forward scans but *easy* during backward ones, whereas checking $\mathfrak{L}_2$ is *easy* during forward scans but *hard* during backward ones. Then, a small 2DFA may just scan forwards until it recognizes a positive instance of $\mathfrak{L}_2$, then turn backwards until it recognizes a positive instance of $\mathfrak{L}_1$, then turn forwards again to accept off $\dashv$. In contrast, a small SDFA is in a tough place: although it can, too, recognize the positive instances of each problem when scanning in the appropriate direction, it cannot compare their positions, since it is forced to keep moving until the next endmarker and, upon reaching it, has no accurate memory of these positions any more.

Following is a hardness propagation lemma for the ordered star. Intuitively, it says that, if $\mathfrak{L}_1$ requires 'many' states during forward scans and $\mathfrak{L}_2$ requires 'many' states during backward scans, then $\mathfrak{L}_1 < \mathfrak{L}_2$ requires 'many' states during multiple scans in both directions.

Lemma 8. *If no* $\cup_{\text{L}}1\text{DFA}$ *with* $\binom{m}{2}$*-state components solves* $\mathfrak{L}_1$ *and no* $\cup_{\text{R}}1\text{DFA}$ *with* $\binom{m}{2}$*-state components solves* $\mathfrak{L}_2$*, then no* m*-state* SDFA *solves* $\mathfrak{L}_1 < \mathfrak{L}_2$*.*

Proof. For the contrapositive, suppose some m-state SDFA M solves $\mathfrak{L}_1 < \mathfrak{L}_2$. Among all strings of #-delimited instances of $\mathcal{L}_1$ and $\mathcal{L}_2$, focus on those where neither problem contributes positive instances and those where exactly one of them does:

$$L := \{\#x_1\#\cdots\#x_l\# \mid (\forall i)(x_i \in \tilde{L}_1 \cup \tilde{L}_2)\ \}$$
$$\tilde{L} := \{\#x_1\#\cdots\#x_l\# \mid (\exists i)(x_i \in L_1 \cup L_2)\ \ \&\ \ \neg(\exists i)(\exists j)(x_i \in L_1\ \&\ x_j \in L_2)\ \}$$

where $\#x_1\#\cdots\#x_l\#$ means $l \geq 0$ and every $x_i \in L_1 \cup \tilde{L}_1 \cup L_2 \cup \tilde{L}_2$. Since M solves $\mathfrak{L}_1 < \mathfrak{L}_2$, we know that M solves $\mathfrak{L} = (L, \tilde{L})$.

Now, let ϑ be any generic string for M over L. As usual, let $A := Q_{\text{LR}}(\vartheta)$ and $B := Q_{\text{RL}}(\vartheta)$ be the two sets of outcomes of ϑ and, for any instance x of $\mathfrak{L}_1$ or $\mathfrak{L}_2$, let $(\alpha_x, \beta_x) := (\alpha_{\vartheta,x\vartheta}, \beta_{\vartheta x,\vartheta})$ be the inner behavior of M on the block $\vartheta x\vartheta$. Using Fact 8, we can get a criterion for checking whether x is positive.

Claim 1. *An instance* x *of* $\mathfrak{L}_1$ *or* $\mathfrak{L}_2$ *is positive iff* (α_x, β_x) *do not permute* (A, B)*.*

Proof. If x is positive, then clearly $\vartheta x\vartheta \in \tilde{L}$. The same holds for every $\vartheta x^{(k)}\vartheta = \vartheta(x\vartheta)^k$ with $k \geq 1$, since the only positive instances are the copies of x and they all come from the same problem. Thus, every infix $x^{(k)}$ is negative relative to $\mathfrak{L}$ (cf. Section 4.3). Hence, (α_x, β_x) do not permute (A, B), by Fact 8.

If x is negative, then clearly $\vartheta x\vartheta \in L$. Thus, the infix $x^{(1)} = x$ is positive relative to $\mathfrak{L}$. Hence, (α_x, β_x) permute (A, B), by Fact 8 again. ⊡

For positive instances, the above criterion is somewhat weak. It says that at least one of α_x and β_x is not permutative, without saying which. It turns out that a stronger criterion is possible for at least one of $\mathfrak{L}_1$ or $\mathfrak{L}_2$, exactly because M can tell the relative placement of their positive instances. Intuitively, a non-permutative α_x means that forward scans by M can 'sense' that x is positive, whereas a non-permutative β_x means the same for backward scans by M.

Claim 2. *At least one of the following statements is true:*
- *for every positive instance* x *of* $\mathfrak{L}_1$*:* α_x *does not permute* A*,*
- *for every positive instance* x *of* $\mathfrak{L}_2$*:* β_x *does not permute* B*.*

Proof. Suppose neither statement is true. Then there exist $x \in L_1$ and $y \in L_2$ such that α_x permutes A and β_y permutes B. Pick $k \geq 1$ so that the two permutations become identities after k iterations:

$$(\alpha_x)^k = \text{id}_A \qquad \text{and} \qquad (\beta_y)^k = \text{id}_B\,.$$

Then $\alpha_{x^{(k)}} \geq (\alpha_x)^k = \text{id}_A$ and $\beta_{y^{(k)}} \geq (\beta_y)^k = \text{id}_B$ (Fact 6), and therefore

$$\alpha_{x^{(k)}} = \text{id}_A \qquad \text{and} \qquad \beta_{y^{(k)}} = \text{id}_B\,,$$

since id_A and id_B are total.

Intuitively, this means that forward scans cannot distinguish between ϑ and $\vartheta x^{(k)}\vartheta$, whereas backward scans cannot distinguish betweeen ϑ and $\vartheta y^{(k)}\vartheta$. Hence, M should be unable to distinguish between the two blocks

$$\vartheta x^{(k)}\vartheta y^{(k)}\vartheta \qquad \text{and} \qquad \vartheta y^{(k)}\vartheta x^{(k)}\vartheta\,, \tag{8}$$

because they should both look like $\vartheta y^{(k)}\vartheta$ during forward scans and like $\vartheta x^{(k)}\vartheta$ during backward scans. If this intuition is correct, then M decides identically on a positive and a negative instance of $\mathfrak{L}_1 < \mathfrak{L}_2$—the desired contradiction.

Indeed, if we calculate the forward part of the inner behavior of M on each of the two blocks in question, we find (using Fact 5 in the first step):

$$\begin{aligned}\alpha_{x^{(k)}\vartheta y^{(k)}} &\geq \alpha_{x^{(k)}} \circ \alpha_{y^{(k)}} = \mathrm{id}_A \circ \alpha_{y^{(k)}} = \alpha_{y^{(k)}}\\ \alpha_{y^{(k)}\vartheta x^{(k)}} &\geq \alpha_{y^{(k)}} \circ \alpha_{x^{(k)}} = \alpha_{y^{(k)}} \circ \mathrm{id}_A = \alpha_{y^{(k)}}\,.\end{aligned}$$

Since M is sweeping, all its inner behaviors consist of total functions. Hence, $\alpha_{y^{(k)}}$ is total, causing $\alpha_{x^{(k)}\vartheta y^{(k)}} = \alpha_{y^{(k)}} = \alpha_{y^{(k)}\vartheta x^{(k)}}$. By this and a symmetric argument for the backward parts, we eventually conclude that

$$(\alpha_{x^{(k)}\vartheta y^{(k)}}, \beta_{x^{(k)}\vartheta y^{(k)}}) = (\alpha_{y^{(k)}}, \beta_{x^{(k)}}) = (\alpha_{y^{(k)}\vartheta x^{(k)}}, \beta_{y^{(k)}\vartheta x^{(k)}})$$

in accordance with our intuition above. It follows that M behaves identically on the two blocks of (8), by a straightforward argument that compares the decompositions of the computations on them relative to their infixes (as in the proof of Lemma 3). ⊡

Now, if the first statement of Claim 2 is true, we can combine it with Claim 1 to arrive at the following criterion for $\mathfrak{L}_1$:

$$\text{an instance } x \text{ of } \mathfrak{L}_1 \text{ is positive iff } \alpha_x \text{ does not permute } A. \tag{9}$$

This leads us to a small-component $\cup_{\text{L}}1\text{DFA}$ for $\mathfrak{L}_1$, as shown in the following.

Claim 3. *Some* $\cup_{\text{L}}1\text{DFA}$ *solves* $\mathfrak{L}_1$ *with* $\binom{m}{2}$*-state components.*

Proof. Let $x \in L_1 \cup \tilde{L}_1$. Since M is sweeping, we know $\alpha_x : A \rightharpoonup Q$ is total. Hence, it can fail to be a permutation in two ways:[7] by not keeping all its values inside A or by not being injective. Therefore, a restatement of (9) is that

$$\begin{aligned}x \in L_1 \iff\ & (\exists p \in A)\big(\alpha_x(p) \notin A\big)\\ & \vee\ (\exists p_1, p_2 \in A)\big(p_1 \neq p_2\ \ \&\ \ \alpha_x(p_1) = \alpha_x(p_2)\big).\end{aligned} \tag{10}$$

The condition $\alpha_x(p) \notin A$ is equivalent to saying that $\textsc{lcomp}_p(x\vartheta)$ hits right into a state outside A. In turn, because this latter computation is simple (even simpler than in Fig. 8a, since M is sweeping), this can be restated as

$$\begin{aligned}&\textsc{lcomp}_p(x) \text{ hits right into a state } q \text{ such that}\\ &\qquad\qquad \textsc{lcomp}_q(\vartheta) \text{ hits right into a state outside } A\,.\end{aligned} \tag{11}$$

Similarly, the condition $\alpha_x(p_1) = \alpha_x(p_2)$ is equivalent to saying that

$$\begin{aligned}&\textsc{lcomp}_{p_1}(x) \text{ and } \textsc{lcomp}_{p_2}(x) \text{ hit right into states } q_1 \text{ and } q_2 \text{ such that}\\ &\qquad\qquad \textsc{lcomp}_{q_1}(\vartheta) \text{ and } \textsc{lcomp}_{q_2}(\vartheta) \text{ hit right into the same state.}\end{aligned} \tag{12}$$

Overall, (10)-(12) describe a way to test $x \in L_1$ by simulating M only on x.

Specifically, for any $p_1, p_2 \in A$ we build a 1DFA M_{p_1,p_2} which checks (12) for p_1, p_2 and also (11) for p_1 and for p_2. On input x, the machine performs a synchronized simulation of both $\textsc{lcomp}_{p_1}(x)$ and $\textsc{lcomp}_{p_2}(x)$. If at any point the two computations are about to enter the same state, the machine hangs. If the right endmarker is ever reached, the machine knows the two states q_1 and q_2 produced, and thus also the states r_1 and r_2 produced by $\textsc{lcomp}_{q_1}(\vartheta)$ and $\textsc{lcomp}_{q_2}(\vartheta)$. Hence, it accepts iff $r_1 \notin A$ or $r_2 \notin A$ or $r_1 = r_2$. Note that this final test is symmetric in r_1 and r_2. Hence, throughout the simulation, the current states of the two computations may be recorded as an unordered pair. Therefore, M_{p_1,p_2} does not need more than $\binom{m}{2}$ states.

Now consider the $\cup_{\text{L}}$1DFA whose $\binom{m}{2}$ components are the 1DFAs M_{p_1,p_2}, where p_1, p_2 range over all unordered pairs of states of M. This $\cup_{\text{L}}$1DFA solves $\mathfrak{L}_1$. ⊡

Symmetrically, if the second statement of Claim 2 is true, then we get a small-component $\cup_{\text{R}}$1DFA for $\mathfrak{L}_2$, and the proof of Lemma 8 is complete. □

7.3. The separation

We are now ready to define the problem separating small 2-reversal 2DFAs from small SDFAs. It is constructed from the core problem $\mathfrak{J}$ of (7) by applying the problem transformations introduced in the last two sections:

$$\mathfrak{J}' := \left(\bigwedge \mathfrak{J}\right) < \left(\bigwedge \mathfrak{J}^{\text{R}}\right) \tag{13}$$

To decode this, note that every instance of $\bigwedge\mathfrak{J}$ is of the form $\#\alpha_1 i_1 \# \cdots \# \alpha_l i_l \#$ where the α are sets, the i are numbers, and the question is whether every set contains the adjacent number. The instances of $\bigwedge\mathfrak{J}^{\text{R}}$ ask the same question, but have the form $\#i_1\alpha_1\# \cdots \#i_l\alpha_l\#$, with numbers before sets. Consequently, the instances of $\mathfrak{J}'$ have the form $*x_1* \cdots *x_l*$ (note the fresh delimiter $*$) where every x is a list of either set-number pairs or number-set pairs. A list of either kind is positive if every set in it contains its adjacent number. The promise is that all positive lists of one kind appear before all positive lists of the other kind. The question is whether no list is positive *or* positive lists of both kinds exist and the number-set ones appear first.

To solve $\mathfrak{J}'$ on a small 2-reversal 2DFA, we work as follows. First, we let M_0 be the h-state 1DFA which solves $\mathfrak{J}^{\text{R}}$. Then we construct a $O(h)$-state 1DFA M_1 which solves $\bigwedge\mathfrak{J}^{\text{R}}$ by performing successive simulations of M_0 and checking if all accept. Finally, we use M_1 as a subroutine in the following algorithm.

1. We scan forwards, ignoring instances of $\bigwedge\mathfrak{J}$ and simulating M_1 on instances of $\bigwedge\mathfrak{J}^{\text{R}}$. If a positive instance is found, we raise a flag $\mathtt{b} = 1$ and go to 2; otherwise, we eventually reach $\dashv$ and go to 2 with $\mathtt{b} = 0$.

2. We scan backwards, ignoring instances of $\bigwedge\mathfrak{J}^{\mathrm{R}}$ and simulating M_1 on (the reverses of) instances of $\bigwedge\mathfrak{J}$. If a positive instance is found, we raise a flag $\mathtt{a} = 1$ and go to 3; otherwise, we eventually reach $\vdash$ and go to 3 with $\mathtt{a} = 0$.
3. We scan forwards until $\dashv$, where we accept iff $\mathtt{a} = \mathtt{b}$.

Clearly, the algorithm performs exactly 2 reversals on every instance of $\mathcal{J}'$, and its three stages require respectively $O(h) + O(h) + O(1) = O(h)$ states. Concerning correctness, we take cases with respect to the values of the flags.

If $\mathtt{a}$=0 & $\mathtt{b}$=0, then each of stages 1 and 2 scanned the entire input and found no positive instance of $\bigwedge\mathfrak{J}^{\mathrm{R}}$ and of $\bigwedge\mathfrak{J}$, respectively. Hence, neither problem constributed positively. Therefore, the input is positive.

If $\mathtt{a}$=1 & $\mathtt{b}$=1, then stages 1 and 2 found two positive instances, one of $\bigwedge\mathfrak{J}^{\mathrm{R}}$ and one of $\bigwedge\mathfrak{J}$, with the former to the right of the latter. So (by the promise), all positive instances of $\bigwedge\mathfrak{J}^{\mathrm{R}}$ appear after those of $\bigwedge\mathfrak{J}$. So, the input is positive.

If $\mathtt{a}$=1 & $\mathtt{b}$=0, then stage 1 scanned the entire input and found no positive instance of $\bigwedge\mathfrak{J}^{\mathrm{R}}$, whereas stage 2 found a positive instance of $\bigwedge\mathfrak{J}$. Hence, only $\bigwedge\mathfrak{J}$ contributed positively. So, the input is negative.

If $\mathtt{a}$=0 & $\mathtt{b}$=1, then stage 1 found a positive instance of $\bigwedge\mathfrak{J}^{\mathrm{R}}$ and, to the left of it, stage 2 found no positive instances of $\bigwedge\mathfrak{J}$. Thus, *either* $\bigwedge\mathfrak{J}$ contributes no positive instances, hence only $\bigwedge\mathfrak{J}^{\mathrm{R}}$ does, *or* $\bigwedge\mathfrak{J}$ contributes positive instances only after the first positive instance of $\bigwedge\mathfrak{J}^{\mathrm{R}}$, hence (by the promise) all its positive instances appear after those of $\bigwedge\mathfrak{J}^{\mathrm{R}}$. Either way, the input is negative.

Overall, the input instance is positive iff $\mathtt{a} = \mathtt{b}$, and the algorithm is correct.

To prove that $\mathfrak{J}'$ is hard for SDFAs, we work as follows. First, we recall that no 1DFA solves $\mathfrak{J}$ with $< 2^h$ states. So, the same holds for $\neg\mathfrak{J}$, since 1DFAs can always be complemented without increasing the number of states. Therefore, no $\cap_{\mathrm{L}}$1DFA solves $\bigvee\neg\mathfrak{J}$ with components of $< 2^h$ states (by Lemma 7). Therefore,

$$\text{no } \cup_{\mathrm{L}}\text{1DFA solves } \bigwedge\mathfrak{J} \text{ with components of } < 2^h - 1 \text{ states} \tag{14}$$

by the second statement of Fact 12 applied on $\neg\bigwedge\mathfrak{J} = \bigvee\neg\mathfrak{J}$. Therefore,

$$\text{no } \cup_{\mathrm{R}}\text{1DFA solves } \bigwedge\mathfrak{J}^{\mathrm{R}} \text{ with components of } < 2^h - 1 \text{ states} \tag{15}$$

by the first statement of Fact 12 applied on $(\bigwedge\mathfrak{J})^{\mathrm{R}} = \bigwedge\mathfrak{J}^{\mathrm{R}}$. Therefore,

$$\text{every SDFA solving } \mathfrak{J}' \text{ needs } 2^{\Omega(h)} \text{ states,}$$

by (14), (15), and Lemma 8 applied on $(\bigwedge\mathfrak{J}) < (\bigwedge\mathfrak{J}^{\mathrm{R}}) = \mathfrak{J}'$.

8. Conclusion

We confirmed the Sakoda-Sipser conjecture in the special case of two-way finite automata which perform sublinearly many reversals, by proving that 2DFAs of this kind must be exponentially large to solve ONE-WAY LIVENESS. We also showed that our theorem does not resolve the full conjecture, because in some cases raising the number of reversals of a 2DFA from sublinear to linear results in exponential savings in size. Finally, we proved that exponential savings in

size are possible even when we raise the number of within-the-input reversals from zero to the smallest possible non-zero number.

All our witnesses were defined over alphabets of exponential size. However, up to polynomial differences, our conclusions remain valid even over the binary alphabet. For example, the binary version of OWL_h where each symbol of Σ_h is encoded into a h^2-bit string in the natural way, still needs $\Omega(2^h)$ states on every 2DFA with few reversals (by the same proof, as all reasoning is on cell boundaries) but only $O(h^2)$ states on a 2NFA with zero reversals. Therefore, the title of this article remains valid even if we change the interpretation of 'small 2FA' from '2FA with few states' to '2FA with short description'.

Theorem 1 says that every 2DFA for OWL_h satisfies

$$r_M(n) \neq o(n) \quad \vee \quad |Q| = \Omega(2^h)\,.$$

It would be interesting to see a proof of the following stronger condition

$$r_M(n) = \Omega(n) \quad \vee \quad |Q| \geq 2^h\,.$$

Another direction for further work is to continue with Research Problem 4 of [11], analyzing the trade-off between size and number of reversals of a 2DFA.

In the broader horizon, two complementary directions are suggested.

The first points, of course, towards the full *Sakoda-Sipser conjecture*. This is perhaps even more inviting now, after the recent tightening of the connections to $\mathsf{L} \vee \mathsf{NL}$ [10]. Specifically, resolving the conjecture can now be seen as a first step, hopefully more tractable than others, towards the conjectures $\mathsf{NLL} \not\subseteq \mathsf{LL/polylog}$ and $\mathsf{NL} \not\subseteq \mathsf{L/poly}$ (in this order), where the classes NL and $\mathsf{L/poly}$ correspond (as usual) to $O(\log n)$ space and $\text{poly}(n)$ advice bits, whereas NLL and $\mathsf{LL/polylog}$ are their counterparts for space $O(\log \log n)$ and advice of length $\text{poly}(\log n)$.

The second direction points towards the full *Sakoda-Sipser analogy*. This refers to the analogy drawn in [2] between the time complexity of Turing machines and the size complexity of 2FAs. In the same way that the $\mathsf{P} \vee \mathsf{NP}$ theory was the first step towards the complexity theory that has been built on Turing machines and time, the $\mathsf{2D} \vee \mathsf{2N}$ theory of [2] can be seen as the first step towards a complexity theory that can be built on 2FAs and size [17]. Developing this theory appears to be a valuable long-term goal.

References

[1] J. I. Seiferas, Untitled manuscript, communicated to M. Sipser (Oct. 1973).

[2] W. J. Sakoda, M. Sipser, Nondeterminism and the size of two-way finite automata, in: Proceedings of the Symposium on the Theory of Computing, 1978, pp. 275–286.

[3] M. Sipser, Lower bounds on the size of sweeping automata, Journal of Computer and System Sciences 21 (2) (1980) 195–202.

[4] H. Leung, Tight lower bounds on the size of sweeping automata, Journal of Computer and System Sciences 63 (3) (2001) 384–393.

[5] J. Hromkovič, G. Schnitger, Nondeterminism versus determinism for two-way finite automata: generalizations of Sipser's separation, in: Proceedings of the International Colloquium on Automata, Languages, and Programming, 2003, pp. 439–451.

[6] C. Kapoutsis, Deterministic moles cannot solve liveness, Journal of Automata, Languages and Combinatorics 12 (1-2) (2007) 215–235.

[7] V. Geffert, C. Mereghetti, G. Pighizzini, Converting two-way nondeterministic unary automata into simpler automata, Theoretical Computer Science 295 (2003) 189–203.

[8] P. Berman, A. Lingas, On complexity of regular languages in terms of finite automata, Report 304, Institute of Computer Science, Polish Academy of Sciences, Warsaw (1977).

[9] V. Geffert, G. Pighizzini, Two-way unary automata versus logarithmic space, Information and Computation 209 (7) (2011) 1016–1025.

[10] C. Kapoutsis, Two-way automata versus logarithmic space, in: Proceedings of the International Computer Science Symposium in Russia, 2011, pp. 197–208.

[11] J. Hromkovič, Descriptional complexity of finite automata: concepts and open problems, Journal of Automata, Languages and Combinatorics 7 (4) (2002) 519–531.

[12] C. Kapoutsis, Small sweeping 2NFAs are not closed under complement, in: Proceedings of the International Colloquium on Automata, Languages, and Programming, 2006, pp. 144–156.

[13] S. Jukna, Extremal Combinatorics, Springer-Verlag, 2001.

[14] S. Micali, Two-way deterministic finite automata are exponentially more succinct than sweeping automata, Information Processing Letters 12 (2) (1981) 103–105.

[15] C. Kapoutsis, Algorithms and lower bounds in finite automata size complexity, PhD Thesis, Massachusetts Institute of Technology (Jun. 2006).

[16] C. Kapoutsis, R. Královič, T. Mömke, On the size complexity of rotating and sweeping automata, in: Proceedings of the International Conference on Developments in Language Theory, 2008, pp. 455–466.

[17] C. Kapoutsis, Size complexity of two-way finite automata, in: Proceedings of the International Conference on Developments in Language Theory, 2009, pp. 47–66.

Notes

(1)Note the difference between (1) and the respective definition of [6, §3.2]: There, $Q_{\text{LR}}(y)$ is the states produced by all possible LR-traversals. Here, $Q_{\text{LR}}(y)$ is only those produced by LR-traversals that are segments of full computations.

(2)Note the difference between the inclusion $Q_{\text{LR}}(yz) \subseteq \alpha_{y,z}[Q_{\text{LR}}(y)]$ and Fact 3 of [6]: There, we know (easily) that all values of $\alpha_{y,z}$ are inside $Q_{\text{LR}}(yz)$ (and cover it), so we call $\alpha_{y,z}$ a 'surjection onto $Q_{\text{LR}}(yz)$'. Here, some values of $\alpha_{y,z}$ may fall outside $Q_{\text{LR}}(yz)$, so we cannot call $\alpha_{y,z}$ a 'surjection onto $Q_{\text{LR}}(yz)$' — we cannot even call it a 'function to $Q_{\text{LR}}(yz)$'. We only know that its values cover $Q_{\text{LR}}(yz)$ —they may also cover parts of $Q \setminus Q_{\text{LR}}(yz)$.

(3)Note a difference from [6, Lemma 3]: There, we required that L is infinitely extensible to the right/left. This is redundant. If L is not infinitely extensible to the right, then LR-generic strings still exist: every $y \in L$ with no right-extensions in L is (vacuously) LR-generic. Similarly in the other direction.

(4)Here the conclusion is the same as [6, Lemma 5], but the reasoning differs. In both cases, we show that the partial map $\alpha_{y,z} : Q_{\text{LR}}(y) \rightharpoonup Q$ is really a bijection to $Q_{\text{LR}}(yz)$; namely, that it is (i) total, (ii) injective, (iii) covers $Q_{\text{LR}}(yz)$, and (iv) stays inside $Q_{\text{LR}}(yz)$. In [6], Fact 3 shows (iii) and (iv), and genericity implies the rest. Here, Fact 4a shows only (iii), and genericity implies the rest.

(5)Our blocks are the same as those of [3]. Note that the *traps* of [6] are blocks, too, but their infixes must always preserve the property L of the generic string.

The intentions of [6] and [3] differ. In [6], we want to force the machine into permutative inner behavior, so as to then prove that it gets lost in the maze. Instead, in [3] (and here) we want to study how the machine changes behavior between infixes that preserve and infixes that purturb the generic string's property, so as to then prove that these changes require many states.

(6)Note the difference from [6, Fact 7]: There, we know that all infixes involved (x, y, and z) preserve L, thus all functions (α_x, α_y, and α_z) are total, and we just need to make sure that their values match. Here, any of the infixes may not preserve L. So, we can only guarantee that $\alpha_z(p)$ is defined when both $\alpha_x(p)$ and $\alpha_y(\alpha_x(p))$ are —and that then its value is as expected. Note that the converse is not necessarily true: if $\alpha_z(p)$ is defined, namely $c_z := \text{COMP}_{p,|\vartheta|+1}(\vartheta x \vartheta y \vartheta)$ hits right into some state r, then c_z certainly crosses the $\vartheta x \vartheta$-$y\vartheta$ boundary, the first time into some state q, and thus $\alpha_x(p)$ is also defined and equals q; however, the suffix of c_z after this first crossing may very well revisit the prefix ϑx, in which case $\alpha_y(\alpha_x(p)) = \alpha_y(q)$ will be undefined.

(7)Note a difference between this construction and the one in the proof of [16, Lemma 8]. There, α_x (is denoted by $\text{LMAP}(\vartheta, x\vartheta)$, and) is easily known to keep all its values inside A. As a result, the 1DFA components in that construction need to test only one of the two conditions that the components in our construction will test.