Journal of Automata, Languages and Combinatorics **12** (2007) 1/2, 215–235

DETERMINISTIC MOLES CANNOT SOLVE LIVENESS [1]

Christos A. Kapoutsis [2]
ETH Zürich, Informationstechnologie und Ausbildung, CAB F13.2
Universitätsstrasse 6, 8092 Zürich, Switzerland
e-mail: christos.kapoutsis@inf.ethz.ch

ABSTRACT

We examine the conjecture that no polynomial can upper bound the increase in the number of states when a one-way nondeterministic finite automaton (1NFA) is converted into an equivalent two-way deterministic finite automaton (2DFA). We study the problem of *liveness*, which admits 1NFAs of polynomial size and is known to defy 2DFAs of polynomial size if and only if the conjecture is true. We focus on *moles*, a restricted class of two-way nondeterministic automata that includes the 1NFAs solving liveness. We show that, in contrast, 2DFA moles cannot solve liveness, irrespective of their size.

Keywords: One-way nondeterministic finite automata, two-way deterministic finite automata, Sakoda-Sipser conjecture, 2D versus 2N, descriptional complexity

1. Introduction

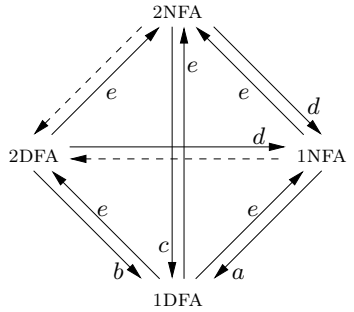
It has been known for a long time [17] that the power of one-way deterministic finite automata (1DFAs) does not increase when they are enhanced with nondeterminism and/or bidirectionality: be they one-way nondeterministic (1NFAs), two-way deterministic (2DFAs), or even two-way nondeterministic (2NFAs), finite automata still fail against non-regular problems. However, this describes the situation only from the point of view of computability.

From the complexity perspective, the extra capabilities do increase the power of 1DFAs, in the sense that against the same problems the enhanced automata occasionally manage to stay exponentially smaller [1, 14, 16]. This observation has initiated a more general and systematic investigation: *when we convert a machine of a particular type into an equivalent machine of a different type, how much 'larger' need the new machine be, in general?* Even the apparently simple world of regular languages hosts some most intriguing instances of this question.

The four types of automata mentioned above define a dozen different conversions (Fig. 1). The famous one is that from 1NFAs to 1DFAs. We know that every n-state

[1] Full version of a submission presented at the 7th Workshop on *Descriptional Complexity of Formal Systems* (Como, Italy, June 30 – July 2, 2005).

[2] This research was carried out while the author was at the Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory.



$a = 2^n - 1$ [17, 14]

$b = n\big(n^n - (n-1)^n\big)$ [11]

$c = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \binom{n}{i}\binom{n}{j}\big(2^i - 1\big)^j$ [11]

$d = \binom{2n}{n+1}$ [11]

$e = n$ [4]

Figure 1: The 12 conversions defined by nondeterminism and bidirectionality, and the known exact trade-offs. Dashed arrows mark the two open problems.

1NFA can be simulated by a 1DFA that has at most $2^n - 1$ states [17]; we also know that, for every n, there exist n-state 1NFAs that have no equivalent 1DFA with fewer than $2^n - 1$ states [14]; we therefore say that the *trade-off* from 1NFAs to 1DFAs is *exactly* $2^n - 1$. Similarly, the exact value of the trade-off for each of the remaining conversions is known [4, 11, 12]. Except two.

1.1. The Problem

It is surprising how little we know about the conversion from 2NFAs to 2DFAs. Not only do we not know the exact value of the associated trade-off, but we cannot even tell whether it is polynomial. The best known upper bound is exponential (by the conversion down to 1DFAs, Fig. 1c) and the best known lower bound is quadratic (by the conversion from just unary 1NFAs [6]).

When he first posed the question, Seiferas [19] conjectured that the trade-off is at least $2^n - 1$ even when the 2NFA being converted is actually one-way, a 1NFA. Given that a 1NFA can always be converted into a 2DFA via the 1NFA-to-1DFA conversion with at most this cost, the conjecture amounts to saying that the best general method for going from 1NFAs to 2DFAs is actually via 1DFAs —a rather impressive claim. It is this problem that we study. To sum up, the exact value of the trade-off from 1NFAs to 2DFAs is conjectured to be $2^n - 1$, equal to its best known upper bound, and much greater than the best known lower bound —the quadratic one mentioned above.

Already in [19], Seiferas proved the conjecture under the restriction that the 2DFAs are *single-pass*, in the sense that they halt as soon as they reach an end-marker. Later, Sipser [20] did the same under the restriction that the 2DFAs are *sweeping*, meaning that they can switch direction only on end-markers —Leung [13] showed this separation holds even on a binary alphabet, as opposed to the exponentially large one of [20]. Recently, Hromkovic and Schnitger [9] established the conjecture for the case when the 2DFAs are *oblivious*, in the sense that they move identically on all inputs of the same length —they also showed the lower bound remains exponential if we relax the restriction to allow a sublinear (in the input length) number of distinct trajectories. None of these theorems resolves the conjecture in its generality, as full 2DFAs can be

exponentially more succinct than each of these restricted variants [19, 20, 2, 15].

Beyond limited bidirectionality, Chrobak [6] disproved the conjecture for the case of *unary* automata, showing that the trade-off is at most $O(n^2)$ —and also at least $\Omega(n^2)$, the best known lower bound mentioned above. For the more general case, when the unary automaton being converted is a 2NFA, Geffert, Mereghetti and Pighizzini [7] have recently established the sub-exponential upper bound $2^{\Theta(\lg^2 n)}$ —their subsequent theorem [8], that the trade-off in the complementation of unary 2NFAs is indeed polynomial, suggests that a polynomial upper bound may actually be possible.

Finally, variations of the problem have appeared. If we demand that the 2DFA *can* decide identically to the simulated 2NFA *no matter what state and input position the latter is started at* (a requirement conceptually stronger than ordinary simulation, but always satisfiable [5]), then the trade-off is at least $2^{\lg^k n}$, for any k [10]. If we demand that the 2DFA decides identically to the simulated 2NFA *only on all polynomially long inputs* (a requirement conceptually weaker than ordinary simulation), then an exponential lower bound would confirm the old belief that nondeterminism is essential in logarithmic-space Turing machines (L≠NL) [3]. Last, if we allow the starting 2NFA to be a *Hennie machine* (a more powerful device, but still not powerful enough to solve non-regular problems), then converting to a 2DFA indeed costs exponentially, but only because converting to a 2NFA already does [4].

1.2. Our Approach

Soon after Seiferas posed the problem, Sakoda and Sipser [18] invested it with a theoretical framework. One of their conclusions was that, in order to determine whether the 1NFA-to-2DFA trade-off is polynomial or not, it is enough to study a particular *complete* language, B_n: if there is a polynomial p and a 2DFA that recognizes B_n with at most $p(n)$ states, then the trade-off is polynomial; otherwise, it is not. More intuitively and concisely, *the trade-off is polynomial iff a small* 2DFA *can solve* B_n.

Before we describe B_n, let us discuss the alphabet Σ_n over which it is defined. This consists of all directed 2-column graphs with n nodes per column and only rightward arrows (Fig. 2a). Note that Σ_n contains 2^{n^2} symbols. An m-long string over Σ_n is naturally viewed as a directed $(m+1)$-column graph (Fig. 2b). In this graph, a node is called *live* if it can be reached from the leftmost column. If at least one of the nodes of the rightmost column is live, then the entire string is also called *live*; otherwise, it is called *dead*. For simplicity, we often omit the direction of the arrows (Fig. 2c). In this undirected representation, liveness amounts to the existence of a path which connects the 0th to the mth column and has exactly m edges.

The language B_n consists of all live strings over Σ_n. It is the property of *liveness*.

One of the (two) reasons why this language is complete is that a 1NFA can recognize it with polynomially many states. In fact, there exists a 1NFA N_n that does so with only n states: At every step, each branch in N_n's computation 'remembers' one of the live nodes of the current column; on reading the next symbol, it finds what arrows depart from that node, chooses one nondeterministically, and follows it. Our approach is inspired by the way this particular solver of B_n works.

More specifically, note that, although at every step N_n reads the entire next symbol,

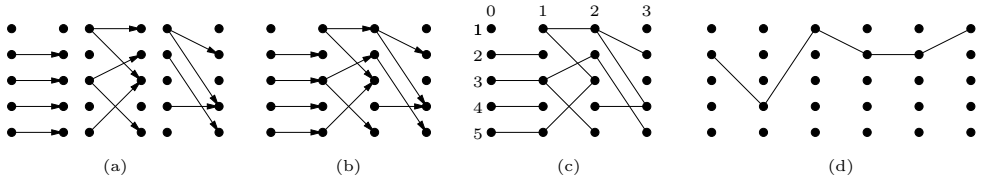


Figure 2: (a) Three symbols in Σ_5. (b) The string they define. (c) The same string, simplified and indexed. (d) A 5-long 2-$\{1, 2, 4\}$-1 path, which is 2-disjoint on itself.

it actually uses only part of the information in it: each branch 'sees' only the arrows leaving the node it is 'focused on' and ignores the rest of the symbol. Put another way, in the 'network of tunnels' defined by the input, N_n behaves like a nondeterministic robot that reads only the index of its current node and the tunnels departing from it. Intuitively, N_n is an n-state one-way nondeterministic *mole*.

It is easy to turn this intuitive description into a formal and general criterion of when a 2NFA is a 'mole' (see Section 2.2). Then, the question about the 1NFA-to-2DFA conversion can be restricted to the world of moles: as before, N_n shows that small 1NFA moles can solve B_n; can small 2DFA moles also succeed? *Is it possible that, after all, small* 2DFA*s* can solve *liveness and the algorithm that achieves this is nothing more than a clever graph exploration?* We give a strong negative answer: *no deterministic mole can recognize B_n, for all $n \geq 5$ and irrespective of size.*

More generally, we view this study as a first step in a qualitatively new direction: instead of studying 2DFAs of *restricted bidirectionality* (single-pass, sweeping, oblivious) but *unrestricted information* (the automata use all current symbol information), we examine 2DFAs of *unrestricted bidirectionality* but of *restricted information.*

The next section formally defines the objects we work with and establishes some of their most basic properties. In Section 3, we introduce some techniques for constructing hard inputs for 2DFAs. The proof of our claim is given in Section 4.

2. The Formal Framework

We write $[n]$ for the set $\{1, 2, \ldots, n\}$. For A and B sets, $|A|$ denotes size; $A \ominus B$ denotes symmetric difference. For f and g functions, $f \circ g$ and fg denote their composition, returning $g\big(f(x)\big)$ for every x, and f^k denotes the k-fold composition of f with itself.

For Σ an alphabet, Σ^* is the set of all finite strings over Σ. If w is a string, $|w|$ is its length. The 'j-th *boundary* of w' is the boundary between its jth and $j+1$st symbols, if $1 \leq j < |w|$; or its leftmost (resp., rightmost) boundary, if $j = 0$ ($j = |w|$). (Fig. 3a.) The string of $k \geq 0$ copies of w is denoted by w^k.

2.1. Finite Automata

We assume the reader is familiar with the intuitive notions of the automata mentioned in the introduction. Here, we define them formally. We start with 2DFAs, which is the most natural model, and present the other types of automata as variations.

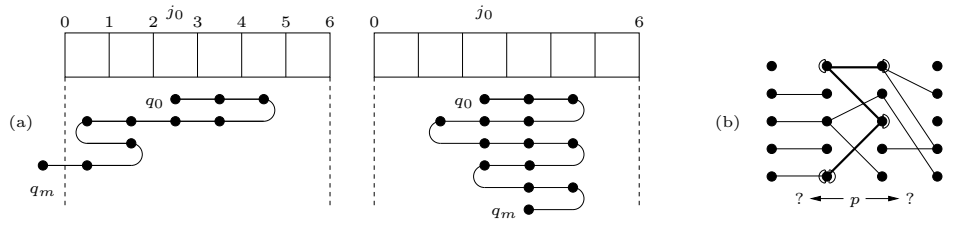


Figure 3: (a) A 6-long string, a computation that hits left, and one that hangs. (b) State p of focus $(5, \mathtt{l})$ is reading the middle symbol: if it moves right, the next focus will be $(1, \mathtt{l})$ or $(3, \mathtt{l})$; if it moves left, the next focus will be $(1, \mathtt{r})$ or $(5, \mathtt{r})$.

A *two-way deterministic finite automaton* (2DFA) is a triple $M = (s, \delta, f)$, where δ is the *transition function*, partially mapping $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to $Q \times \{\mathtt{l}, \mathtt{r}\}$, for some set Q of *states*, some alphabet Σ, two end-marking symbols $\vdash, \dashv \notin \Sigma$, and two direction tags $\mathtt{l}$ and $\mathtt{r}$, while s and f are the *start* and *final* states. We insist that δ never violates an end-marker: on a pair of the form $(\cdot, \vdash)$ (resp., of the form $(\cdot, \dashv)$), it can never return a pair of the form $(\cdot, \mathtt{l})$ (of the form $(\cdot, \mathtt{r})$).

Typically, an input $w \in \Sigma^*$ is presented to M surrounded by the end-markers and the computation starts at s and on $\vdash$. However, many other possibilities exist: for any w, j, and p, the *computation of M when started at state p on the j-th symbol of the string w* (note that w may or may not be end-marked) is the unique sequence

$$\textsc{comp}_{M,p,j}(w) := \big((q_t, j_t)\big)_{0 \le t \le m}$$

where $(q_0, j_0) = (p, j)$; $0 \le m \le \infty$; every pair is derived from its predecessor via δ and w; every pair is within w ($1 \le j_t \le |w|$), except possibly for the last one; and the last pair is within w iff δ is undefined on the corresponding state and symbol. We say (q_t, j_t) is the *t-th point* and m the *length* of this computation. If $m = \infty$, the computation *loops*. Otherwise, it *hits left into q_m*, if $j_m = 0$; or *hangs at q_m*, if $1 \le j_m \le |w|$; or *hits right into q_m*, if $j_m = |w| + 1$ (Fig. 3a). When $j = 1$ (resp., $j = |w|$) we get the *left* (*right*) *computation of M from p on w*:

$$\textsc{lcomp}_{M,p}(w) := \textsc{comp}_{M,p,1}(w) \quad \text{and} \quad \textsc{rcomp}_{M,p}(w) := \textsc{comp}_{M,p,|w|}(w).$$

We say M accepts $w \in \Sigma^*$ iff the computation $\textsc{lcomp}_{M,s}(\vdash w \dashv)$ hangs at f. The *behavior of M on w* is the partial mapping γ_w from $Q \times \{\mathtt{l}, \mathtt{r}\}$ to $Q \times \{\mathtt{l}, \mathtt{r}\}$ that encodes all possible 'entry-exit pairs' as M computes on w: for every $p \in Q$,

$$\gamma_w(p, \mathtt{l}) := \begin{cases} (q, \mathtt{l}) & \text{if } \textsc{lcomp}_{M,p}(w) \text{ hits left into } q, \\ (q, \mathtt{r}) & \text{if } \textsc{lcomp}_{M,p}(w) \text{ hits right into } q, \\ \text{undefined} & \text{if } \textsc{lcomp}_{M,p}(w) \text{ loops or hangs,} \end{cases}$$

while the value $\gamma_w(p, \mathtt{r})$ is defined analogously, with RCOMP instead of LCOMP.

If M is allowed more than one next move at each step, we say that it is *nondeterministic* (a 2NFA). Formally, this means that δ *totally* maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$

to the *powerset* of $Q \times \{\mathtt{l}, \mathtt{r}\}$ and implies that $C = \text{COMP}_{M,p,j}(w)$ is now a *set* of computations. Then, M accepts $w \in \Sigma^*$ iff some $c \in \text{LCOMP}_{M,s}(\vdash w \dashv)$ hangs at f.

If δ never returns pairs of the form $(\cdot, \mathtt{l})$, we say M is *one-way* (a 1DFA or 1NFA).

2.2. Moles

We now restrict our attention to the alphabet Σ_n and to 2NFAs defined over it.

To refer to a symbol of Σ_n, we list its arrows in brackets: e. g., the rightmost symbol in Fig. 2a is `[12,14,25,44]`. The symbol `[]` containing no arrows is called *the empty symbol*. For a string $x \in \Sigma_n^*$, we define the set of its nodes

$$V_x := \{(i, j) \mid i \in [n] \ \ \& \ \ 0 \le j \le |x|\}$$

(Fig. 2c). The *left-degree* of a node $(i, j) \in V_x$ is the number of its neighbors on the column to its left (column $j - 1$), or 0 if $j = 0$; similarly for its *right-degree*. If x has exactly $|x|$ edges that form 1 live path, we say x is a *path* (Fig. 2d); for $i_{\text{L}}, i_{\text{R}} \in I \subseteq [n]$, we say x is a i_{L}-I-i_{R} *path* if this one live path connects the i_{L}th leftmost node to the i_{R}th rightmost node and visits only nodes with indices in I. If $y \in \Sigma_n^*$, then $x \cup y$ is the unique string of length $\max(|x|, |y|)$ that has all edges of x, all edges of y, and no other edges. For $k \ge 0$, we say y is *k-disjoint* on x if in $x \cup (\mathtt{[]}^k y)$ the edges from x and from y meet at *no* node (Fig. 2d, Fig. 5c).

To define when a 2NFA over Σ_n is a mole, we need a way of describing the notion of a state 'focusing on' some particular node of the current symbol. We define a *focus* to be any pair $(i, s) \in [n] \times \{\mathtt{l}, \mathtt{r}\}$ of *index* and *side*. We write $\overline{s}$ for the side opposite s. The *(i, s)th node* of a string x is the ith node of its leftmost (resp., rightmost) column, if $s = \mathtt{l}$ (if $s = \mathtt{r}$). The connected component of that node in the graph implied by x is called the *(i, s)th component* of x. By $x \restriction (i, s)$ we denote the unique string that has the length of x, all edges of the (i, s)th component of x, and no other edges.

A 2NFA is a mole if each state p of it can be assigned a focus (i_p, s_p) so that, whenever at p, the automaton behaves like a mole located on the (i_p, s_p)th node of the current symbol and facing $\overline{s_p}$: (i) it can 'see' only the component of that node, and (ii) it can 'move' only to nodes in that same component. More carefully:

Definition 1 *Let $M = (\cdot, \delta, \cdot)$ be a* 2NFA *over a set of states Q and the alphabet Σ_n. An* assignment of foci for M *is any mapping $\varphi : Q \to [n] \times \{\mathtt{l}, \mathtt{r}\}$ such that, for any states $p, q \in Q$, symbol $a \in \Sigma_n$, and side $s \in \{\mathtt{l}, \mathtt{r}\}$: whenever M is at p reading a,*

(i) *its next move depends only on the component containing the node which p is focused on: $\delta(p, a) = \delta(p,\, a \restriction \varphi(p))$,*

(ii) *its next state and position can only be such that the new focused node belongs to the same connected component as the node which p is focused on:*

$$\delta(p, a) \ni (q, s) \implies (\exists i \in [n])\big(\varphi(q) = (i, \overline{s}) \ \ \& \ \ a \restriction (i, s) = a \restriction \varphi(p)\big).$$

We say $\varphi(p)$ is the focus of p. *If an assignment of foci for M exists, M is a* mole.

To understand Condition (ii), consider as an example the case $s = \mathtt{r}$ (see also Fig. 3b): If p on a moves *right* into q, then in the new position q must focus on the *left* column ($\varphi(q) = (\cdot, \overline{s}) = (\cdot, \mathtt{l})$), the one shared with the previous position. Moreover, if in this column q focuses on the ith node ($\varphi(q) = (i, \mathtt{l})$), then in the previous position

this node (now the ith node of the *right* column) must belong to the same connected component as the node which p focused on ($a \restriction (i, \mathtt{r}) = a \restriction \varphi(p)$).

We note that the 1NFA N_n from Section 1.2 clearly satisfies Definition 1.

2.3. Mazes

What makes moles so weak is of course the fact that, as they move through the input, they can only observe the part of the graph directly connected to their current location. The rest of the graph is not observable, even if it occupies the same symbols as the observable part, and therefore does not affect the computation. Lemma 1 below turns this intuition into a clean fact that can be used in proofs. Before stating it, we need to talk about mazes and how moles compute on them and their compositions.

Intuitively, a maze is any string on which some nodes have been designated as 'entry-exit gates' for moles (Fig. 5d). More carefully, for $x \in \Sigma^*$, let $V_x^0 \subseteq V_x$ consist of every node that has exactly one of its two degrees equal to 0 (and can thus serve as a gate). A *maze* on x is any pair (x, X) where $X \subseteq V_x^0$.

The computation of a mole on a maze is the same object as the computation of any 2NFA on any string, with the extra condition that it 'starts by entering a gate' and 'if it exits a gate, it ends immediately'. Formally, let $\chi = (x, X)$ be a maze, $u = (i, j) \in X$ a gate with 0-degree side s, and p a state of a mole M with focus $\varphi(p) = (i, s)$. Then, *the computation* $\textsc{comp}_{M,p,u}(\chi)$ *of* M *on* χ *from* p *and* u (note the overloading of operator COMP) is *a prefix of* either $\textsc{comp}_{M,p,j+1}(x)$ (if $s = \mathtt{l}$) or $\textsc{comp}_{M,p,j}(x)$ (if $s = \mathtt{r}$). The prefix ends the first time (if ever) it reaches a point (q_t, j_t) where the focus $\varphi(q_t) = (i_t, s_t)$ is on a gate with 0-degree side $\overline{s_t}$. Note that x may contain nodes that have degree 0 on one of their two sides but are not gates; the computation may visit the 0-degree side of these nodes without having to end.

To compose two mazes means to draw their strings on top of each other and then discard all coinciding gates (Fig. 5e). More carefully, mazes $\chi = (x, X)$ and $\psi = (y, Y)$ are *composable* iff $|x| = |y|$ (so that $V_x = V_y = V$) and their graphs intersect only at gates and only appropriately: every $v \in V$, *either* has both its degrees equal to 0 in at least one of x, y; *or* is a gate in both mazes, with a different 0-degree side in each of them. If χ, ψ are composable, then their *composition* is the pair $\chi \circ \psi = (x \cup y, X \ominus Y)$. It should be clear that the composition is also a maze.

Note that, by the conditions of composability, in each symbol of $x \cup y$ every non-empty connected component comes entirely from exactly one of x or y. Hence, when a mole reads a symbol, its next step can only depend on exactly one of x or y. Generalizing this observation, we can prove the following.

Lemma 1 *Let* χ *and* ψ *be as above, and* $\omega = \chi \circ \psi$ *be their composition. Consider a computation* $c = \textsc{comp}_{M,p,u}(\chi \circ \psi)$ *of a mole* M *from a gate* $u \in X \ominus Y$ *that happens to come from* X*. A unique list of computations* $c_1, c_2, \ldots$ *exists, such that:*

- *each* c_t *is a computation of* M *on* χ *(resp., on* ψ*) iff* t *is odd (even);*
- c_1 *starts from* p *and* u*; each* c_{t+1} *starts from the state and gate where* c_t *ends;*
- *if we remove the first point of each* c_t *after* c_1 *and then concatenate, we get* c*.*

Put another way, if we can decompose a maze ω into two mazes χ and ψ, then any computation c of a mole on ω can be uniquely decomposed into 'subcomputations' $c_1, c_2, \ldots$ that alternate between χ and ψ. We say these computations are the *fragments* of c with respect to the decomposition $\omega = \chi \circ \psi$. Clearly, *either* all fragments are finite, and then their list is infinite iff c is; *or* not all fragments are finite, in which case their list is finite and the only infinite fragment is the last one. Note that a different decomposition of ω leads to a different decomposition of c.

3. Hard Inputs

In Section 4 we will fix an arbitrary deterministic mole and prove that it fails against liveness. To this end, we will construct inputs on which the automaton decides incorrectly. Those fatally hard strings will be extremely long. However, we will build them out of other, much shorter (but still very long) strings, which already strain the ability of the automaton to process the information on its tape. In this section we describe those shorter strings. We start with inputs which can be built for any 2DFA and later (Section 3.4) focus on inputs that can be built particularly for deterministic moles. So, fix M to be an arbitrary 2DFA over state set Q and alphabet Σ.

3.1. Dilemmas

Consider a property $T \subseteq \Sigma^*$ of the strings over Σ, and assume that it is *infinitely extensible to the right*, in the sense that every string that has the property can be right-extended into a strictly longer one that also has it: $(\forall y \in T)(\exists z)(|z| \neq 0 \ \& \ yz \in T)$.

For any $y \in T$, we can perform the following experiment. For each $p \in Q$, we examine the computation $\textsc{lcomp}_{M,p}(y)$ and check if it *hits right*: if it does, we set a bit $a_{y,p}$ to 1; otherwise, the computation *hangs*, *loops*, or *hits left*, and $a_{y,p}$ is set to 0. In the end, we build the bit-vector $a_y = (a_{y,p})_{p \in Q}$. This is our outcome.

How does the outcome change if we right-extend y into some $yz \in T$? How do a_y and a_{yz} compare? For every p, clearly $\textsc{lcomp}_{M,p}(y)$ is a prefix of $\textsc{lcomp}_{M,p}(yz)$. So, if the first computation hits left, loops, or hangs, so does the second one; but if the first one hits right, there is no guarantee what the second computation does. Hence, all bits in a_y that are 0 keep the same value in a_{yz}; but a bit which is 1 may turn into a 0. Overall, if "$\geq$" is the natural component-wise order, we have the following.

Fact 1 *For all $y, yz \in T$: $a_y \geq a_{yz}$.*

What happens to the outcome of the experiment if we further right-extend y into $yzz' \in T$? And then into $yzz'z'' \in T$? While y is infinitely right-extensible inside T, the outcome may decrease only finitely many times. Obviously then, from some point on it must stop changing. When this happens, the extension of y that we have arrived at is a very useful tool. The following definition and lemma talk about it formally.

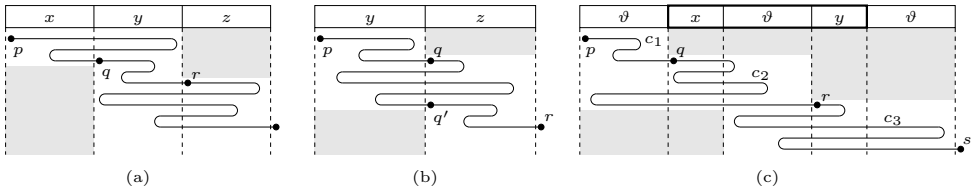


Figure 4: Computations of a 2DFA under dilemmas, generic strings, and traps (see text); each gray region indicates the first or the last crossing of some boundary.

Definition 2 *Let $T \subseteq \Sigma^*$. An* LR-dilemma *over T is any $y \in T$ such that*[3]

$$(\forall yz \in T)(\forall p \in Q)\big[\text{LCOMP}_{M,p}(y) \textit{ hits right} \iff \text{LCOMP}_{M,p}(yz) \textit{ hits right}\big].$$

An RL-dilemma *over T can be defined symmetrically, on left-extensions and* RCOMP.

Lemma 2 *Suppose $T \subseteq \Sigma^*$. If T is non-empty and infinitely extensible to the right (resp., left), then there exist* LR*-dilemmas over T (*RL*-dilemmas over T).*

In [19], dilemmas are called "blocking strings". We now explain these names.

Fact 2 *Assume $x \in \Sigma^*$, y is an* LR*-dilemma over T, $yz \in T$, and that some computation $c = \text{LCOMP}_{M,p}(xyz)$ crosses the xy-z boundary. After the first such crossing, c never visits x again and it eventually hits right.*

Proof. Consider the first time c crosses the xy-z boundary (Fig. 4a). Let r be the state resulting from this crossing, and q the state resulting from the last crossing of the x-yz boundary before that. Then, the computation between these two crossings is $\text{LCOMP}_{M,q}(y)$ and hits right (into r). Since y is an LR-dilemma over T and z does not spoil the property ($yz \in T$), we know that $\text{LCOMP}_{M,q}(yz)$ also hits right. But this computation is a suffix of c. So, c also hits right. Moreover, after crossing the xy-z boundary, it never visits x again. □

In total, once the computation crosses the xy-z boundary, it is restricted inside yz and forced to eventually hit right. Put another way, when M enters y, it faces a 'dilemma': *either* it will stay forever inside xy, never crossing the xy-z boundary; *or* it will cross it, but then also hit right without visiting x again. In effect, y 'blocks' M from returning to x after having seen z —and 'locks' it into hitting right. In yet other words, y makes sure that every left computation of M on xyz that hits left, hangs, or loops does so inside xy, before making it to z.

[3]Note that the displayed condition is the same as $(\forall yz \in T)(a_y = a_{yz})$; but rather more informative. Also note that the "$\Longleftarrow$" part of the equivalence there is given, by Fact 1. What is important is the "$\Longrightarrow$" part: on every extension in T, the computation will keep hitting right.

3.2. Generic Strings

Consider again a property $T \subseteq \Sigma^*$ which is infinitely extensible to the right. For each $y \in T$, we can define the set of states that can be produced on the rightmost boundary of y by left computations:

$$Q_{\text{LR}}(y) := \big\{q \in Q \mid (\exists p \in Q)\big(\text{LCOMP}_{M,p}(y) \text{ hits right into } q\big)\big\}.$$

How does this set change if we extend y into $yz \in T$? How does it compare to $Q_{\text{LR}}(yz)$?

Consider the function $\alpha_{y,z}$, defined as follows (Fig. 4b): for each $q \in Q_{\text{LR}}(y)$, the computation $\text{COMP}_{M,q,|y|+1}(yz)$ is examined; if it hits right into some state r, then $\alpha_{y,z}(q) := r$; otherwise, it hits left, loops, or hangs, and $\alpha_{y,z}(q)$ is left undefined.

Note that the values of $\alpha_{y,z}$ are all in $Q_{\text{LR}}(yz)$: If r is such a value, then $r = \alpha_{y,z}(q)$ for some $q \in Q_{\text{LR}}(y)$. Hence, the computation $\text{COMP}_{M,q,|y|+1}(yz)$ hits right into r and some computation $\text{LCOMP}_{M,p}(y)$ hits right into q. Combining the two, we get the computation $\text{LCOMP}_{M,p}(yz)$, that hits right into r. Hence, $r \in Q_{\text{LR}}(yz)$.

Moreover, the values of $\alpha_{y,z}$ cover $Q_{\text{LR}}(yz)$: If $r \in Q_{\text{LR}}(yz)$, then some computation $c = \text{LCOMP}_{M,p}(yz)$ hits right into r. We know c crosses the y-z boundary, so let q be the state produced by the first such crossing. The computation before this crossing is $\text{LCOMP}_{M,p}(y)$ and hits right into q, so $q \in Q_{\text{LR}}(y)$. The computation after the crossing is $\text{COMP}_{M,q,|y|+1}(yz)$ and, as a suffix of c, hits right into r. Therefore, $\alpha_{y,z}(q) = r$.

Overall, $\alpha_{y,z}$ is a *partial surjection* from $Q_{\text{LR}}(y)$ to $Q_{\text{LR}}(yz)$. This clearly implies its domain has enough elements to cover the range, so we know $|Q_{\text{LR}}(y)| \geq |Q_{\text{LR}}(yz)|$.

The next fact summarizes our findings. Analogously to $Q_{\text{LR}}(y)$, the set $Q_{\text{RL}}(z)$ consists of all states that can be produced on the leftmost boundary of z by right computations. Clearly, the symmetric arguments apply. Note that these involve a partial surjection $\beta_{y,z}$ from $Q_{\text{RL}}(z)$ to $Q_{\text{RL}}(yz)$, defined analogously to $\alpha_{y,z}$.

Fact 3 *For $y, yz \in T$, the function $\alpha_{y,z}$ partially surjects $Q_{\text{LR}}(y)$ to $Q_{\text{LR}}(yz)$; hence $|Q_{\text{LR}}(y)| \geq |Q_{\text{LR}}(yz)|$. Similarly, in the opposite direction, if $yz, z \in T$ then the function $\beta_{y,z}$ partially surjects $Q_{\text{RL}}(z)$ to $Q_{\text{RL}}(yz)$; hence $|Q_{\text{RL}}(yz)| \leq |Q_{\text{RL}}(z)|$.*

As in Section 3.1, we now ask what happens to the size of the set $Q_{\text{LR}}(y)$ as we keep right-extending y inside T. Although y is infinitely right-extensible, the size of the set can decrease only finitely many times. Hence, from some point on it must stop changing. When this happens, we have arrived at another useful tool.

Definition 3 *Let $T \subseteq \Sigma^*$. A string y is* LR-generic *over T if $y \in T$ and*[4]

$$(\forall yz \in T)\big[|Q_{\text{LR}}(y)| = |Q_{\text{LR}}(yz)|\big].$$

An RL-generic *string over T can be defined symmetrically, on left-extensions and Q_{RL}. A string that is simultaneously* LR-*generic and* RL-*generic over T is called* generic.

Lemma 3 *Suppose $T \subseteq \Sigma^*$. If T is non-empty and infinitely extensible to the right (resp., left), then there exist* LR-*generic strings over T (*RL-*generic strings over T). If y_{L} is* LR-*generic and y_{R} is* RL-*generic, then every string $y_{\text{L}} z y_{\text{R}} \in T$ is generic.*

[4]Note that the "$\geq$" part of the displayed equality $|Q_{\text{LR}}(y)| = |Q_{\text{LR}}(yz)|$ is given, by Fact 3. What is important is the "$\leq$" part: on every extension in T, the set will manage to stay as large.

Proof. For the last claim, we simply note that every right-extension of an LR-generic string inside T is also LR-generic, and the same is true in the other direction. □

Generic strings were introduced in [20], for sweeping automata. As we next show, they strengthen dilemmas. The proof is immediate after the following alternative characterizations of the two classes of strings, in terms of the functions $\alpha_{y,z}$ and $\beta_{y,z}$.

Lemma 4 *Suppose $y \in T \subseteq \Sigma^*$. Then y is an* LR-dilemma *over T iff for all $yz \in T$ the function $\alpha_{y,z}$ is* total. *Similarly for any $z \in T$ and for* RL *and $\beta_{y,z}$.*

Proof. For the forward direction, assume y is an LR-dilemma over T. Consider any $yz \in T$ and any $q \in Q_{\text{LR}}(y)$. (Fig. 4b.) Let $c = \text{LCOMP}_{M,p}(y)$ be a computation that hits right into q. We know c is a prefix of $d = \text{LCOMP}_{M,p}(yz)$. So, d crosses the y-z boundary (the first such crossing is into q), and hence hits right (by Fact 2). Therefore, its suffix $\text{COMP}_{M,q,|y|+1}(yz)$ hits right, too. This implies $\alpha_{y,z}(q)$ is defined.

For the reverse direction, fix $y \in T$ and suppose $\alpha_{y,z}$ is total for all $yz \in T$. Consider any such yz, any $p \in Q$, and assume $c = \text{LCOMP}_{M,p}(y)$ hits right into some state q. (Fig. 4b.) Then $q \in Q_{\text{LR}}(y)$. Therefore, $\alpha_{y,z}(q)$ is defined. This implies $c' = \text{COMP}_{M,q,|y|+1}(yz)$ hits right. Combining c and c', we get the computation $d = \text{LCOMP}_{M,p}(yz)$. As c' is a suffix of d, we know d hits right as well. □

Lemma 5 *Suppose $y \in T \subseteq \Sigma^*$. Then y is* LR-generic *over T iff for all $yz \in T$ the function $\alpha_{y,z}$ is* total and bijective. *Similarly for any $z \in T$ and for* RL *and $\beta_{y,z}$.*

Proof. For the forward direction, say y is LR-generic and pick any $yz \in T$. We already know $\alpha_{y,z}$ is a partial surjection from $Q_{\text{LR}}(y)$ to $Q_{\text{LR}}(yz)$. Since y is LR-generic, we also know the two sets have the same size. So, $\alpha_{y,z}$ must be total and injective. □

Intuitively, a dilemma guarantees that the computations that manage to survive through it will also survive through every extension that preserves the property. A generic string guarantees that, in addition, the computations will keep exiting each extension into different states.

Lemma 6 *Let $T \subseteq \Sigma^*$. Over T, every* LR*-generic string is an* LR*-dilemma and every* LR*-dilemma can be right-extended into an* LR*-generic string. Similarly for* RL*.*

Proof. Lemmata 4 and 5 prove the first claim. For the second claim, we simply note that *every* string in T can be right-extended into LR-generic strings. □

Before moving on, we prove a last fact about the operators Q_{LR} and Q_{RL}.

Fact 4 *For all $y, z \in T$: $Q_{\text{LR}}(yz) \subseteq Q_{\text{LR}}(z)$ and $Q_{\text{RL}}(y) \supseteq Q_{\text{RL}}(yz)$.*

Proof. We prove the first containment. Consider any $r \in Q_{\text{LR}}(yz)$ and any computation $d = \text{LCOMP}_{M,p}(yz)$ that hits right into r. (Fig. 4b.) We know d crosses the y-z boundary. Let q' be the state produced by the *last* such crossing. The computation $\text{LCOMP}_{M,q'}(z)$ is a suffix of d, and therefore also hits right into r. So, $r \in Q_{\text{LR}}(z)$. □

3.3. Traps

Consider a property $T \subseteq \Sigma^*$ which is infinitely extensible in either direction and closed under concatenation. For this section, fix ϑ as a generic string over T, and let

$$L := Q_{\text{RL}}(\vartheta), \qquad R := Q_{\text{LR}}(\vartheta),$$

denote the sets of states producible on the leftmost and rightmost boundary of ϑ by traversing it. Note that, by Lemma 6, we know ϑ is both an LR- and an RL-dilemma.

A *trap* (on ϑ) is any string of the form $\vartheta x \vartheta$, where $x \in T$ is the *infix*.

By Lemma 3 and the closure of T under concatenation, traps are still generic strings. However, they further restrict M's freedom: By Lemma 5, the function $\alpha_{\vartheta,x\vartheta}$ is a total bijection from $Q_{\text{LR}}(\vartheta) = R$ to $Q_{\text{LR}}(\vartheta x \vartheta)$. Since $Q_{\text{LR}}(\vartheta x \vartheta) \subseteq R$ (by Fact 4), $\alpha_{\vartheta,x\vartheta}$ is a total bijection from R to a subset of R. Clearly, this is possible only if this subset is R itself. So, $\alpha_{\vartheta,x\vartheta}$ simply permutes R. Similarly, $\beta_{\vartheta x,\vartheta}$ permutes L, and we proved the next fact. Note that, with ϑ fixed, we can refer to the two permutations associated with $\vartheta x \vartheta$ only through the infix x, as α_x and β_x.

Fact 5 *For all $x \in T$: α_x permutes R and β_x permutes L.*

Intuitively, in each direction, the computations that manage to cross the first copy of ϑ eventually cross the entire trap; but, after this first copy, they collectively do nothing more than simply permute the set of states that they have already produced. As we now show, the two permutations fully describe the behavior[5] of M on the trap.

Fact 6 *For all infixes $x, y \in T$: $(\alpha_x, \beta_x) = (\alpha_y, \beta_y) \implies \gamma_{\vartheta x \vartheta} = \gamma_{\vartheta y \vartheta}$.*

Proof. Suppose $(\alpha_x, \beta_x) = (\alpha_y, \beta_y)$ and consider any $p \in Q$. We show $\gamma_{\vartheta x \vartheta}$ and $\gamma_{\vartheta y \vartheta}$ agree on $(p, \mathtt{l})$ —the proof for $(p, \mathtt{r})$ is similar. We examine the computations $c_x := \text{LCOMP}_{M,p}(\vartheta x \vartheta)$ and $c_y := \text{LCOMP}_{M,p}(\vartheta y \vartheta)$. Clearly, these behave identically up to the first crossing of the 'critical' boundary between ϑ and $x\vartheta$ or $y\vartheta$. *If one of them hits left, loops, or hangs*, it does so inside ϑ (since ϑ is an LR-dilemma) without crossing the critical boundary; so, the other computation behaves identically, thus $\gamma_{\vartheta x \vartheta}(p, \mathtt{l}) = \gamma_{\vartheta y \vartheta}(p, \mathtt{l})$. *If one of them hits right*, then it crosses the critical boundary into some state q and so does the other one; but then they both hit right, into the same state $r := \alpha_x(q) = \alpha_y(q)$, so $\gamma_{\vartheta x \vartheta}(p, \mathtt{l}) = \gamma_{\vartheta y \vartheta}(p, \mathtt{l}) = (r, \mathtt{r})$. □

We call (α_x, β_x) the *inner-behavior* of M on the trap $\vartheta x \vartheta$, to distinguish it from $\gamma_{\vartheta x \vartheta}$.

An interesting case arises when ϑ is an infix of the infix itself. Then the inner-behavior of M on the trap can be deduced from its inner-behavior on the traps that are induced by the other two pieces of the infix.

Fact 7 *Suppose $x, y \in T$ and $z = x\vartheta y$. Then $(\alpha_z, \beta_z) = (\alpha_x \circ \alpha_y, \beta_y \circ \beta_x)$.*

Proof. To show that $\alpha_z = \alpha_x \circ \alpha_y$ (the argument for $\beta_z = \beta_y \circ \beta_x$ is similar), we pick an arbitrary $q \in R$ and show that $\alpha_z(q) = \alpha_y\big(\alpha_x(q)\big)$. (Fig. 4c.) We know q is produced

[5] Recall the definition of behavior, from Section 2.1 (p. 219).

by some right-hitting left computation on ϑ, say $c_1 := \textsc{lcomp}_{M,p}(\vartheta)$ for some state p. Since ϑ is an LR-dilemma over T and $\vartheta z\vartheta \in T$, we know $c := \textsc{lcomp}_{M,p}(\vartheta z\vartheta)$ also hits right, into some state s. Therefore, $\alpha_z(q) = s$. Before hitting right, c surely crosses the $\vartheta x\vartheta$-$y\vartheta$ boundary; let r be the state produced by the first such crossing. Clearly, the computation $c_2 := \textsc{comp}_{M,q,|\vartheta|+1}(\vartheta x\vartheta)$ hits right into r, and hence $\alpha_x(q) = r$. Moreover, the suffix of c after the first crossing of the $\vartheta x\vartheta$-$y\vartheta$ boundary is $c_3 := \textsc{comp}_{M,r,|\vartheta x\vartheta|+1}(\vartheta x\vartheta y\vartheta)$ and obviously hits right into s. However, since ϑ is an LR-dilemma over T and $\vartheta y\vartheta \in T$, we know c_3 never visits the prefix ϑx. Hence, it can also be written as $c_3 = \textsc{comp}_{M,r,|\vartheta|+1}(\vartheta y\vartheta)$. Since it hits right into s, we conclude that $\alpha_y(r) = s$. Overall, $\alpha_z(q) = s = \alpha_y(r) = \alpha_y\big(\alpha_x(q)\big)$. □

An obvious generalization holds when the infix contains multiple copies of ϑ. In a particular case of interest, the infix consists of several ϑ-separated copies of some $x \in T$. Specifically, for any $k \geq 1$, we define $x^{(k)} := x(\vartheta x)^{k-1}$ and prove the following.

Fact 8 *For any $x \in T$ and any $k \geq 1$: $(\alpha_{x^{(k)}}, \beta_{x^{(k)}}) = \big((\alpha_x)^k, (\beta_x)^k\big)$.*

3.4. Hard Inputs to Deterministic Moles

We now assume that the 2DFA M of the previous sections is defined over Σ_n and that it is actually a mole. We will design inputs on which M misses a significant amount of information. All these inputs are going to be paths (cf. Section 2.2).

We fix some $I \subseteq [n]$ and $i \in I$, and consider the set $\Pi \subseteq \Sigma_n^*$ of all i-I-i paths. Clearly, Π is non-empty, infinitely extensible in both directions, and closed under concatenation. Hence, by Lemma 3, generic strings over Π exist. We fix ϑ to be one, and let $\kappa := |\vartheta|$. We also set $L := Q_{\mathrm{RL}}(\vartheta)$, $R := Q_{\mathrm{LR}}(\vartheta)$, and let $\mu := \mathrm{lcm}(|L|!, |R|!)$ be the least common multiple of the sizes of the corresponding permutation groups.

For every length $l \geq 1$, we consider all traps (on ϑ) with infixes of length l and collect into a set Ω_l all inner-behaviors that M exhibits on these traps:

$$\Omega_l := \{(\alpha_x, \beta_x) \mid x \text{ is an } i\text{-}I\text{-}i \text{ path of length } l\}.$$

As shown in the next fact, every inner-behavior that can be induced by an l-long infix can also be induced by an infix of length $l + 2\mu(l + \kappa)$. The subsequent fact explains that sometimes the converse is also true.

Fact 9 *For every $l \geq 1$: $\Omega_l \subseteq \Omega_{l+2\mu(l+\kappa)}$.*

Proof. Pick any behavior $(\alpha, \beta) \in \Omega_l$. We know that some l-long infix $x \in \Pi$ induces this behavior, namely $(\alpha, \beta) = (\alpha_x, \beta_x)$. Consider the path $x^{(2\mu+1)} = x^{(\mu)}\vartheta x\vartheta x^{(\mu)}$. This is also in Π and of length $(2\mu + 1)l + 2\mu\kappa = l + 2\mu(l + \kappa)$. Moreover, by Fact 8 and the selection of μ, we know that this path induces the behavior $(\alpha_x^{2\mu+1}, \beta_x^{2\mu+1}) = \big((\alpha_x)^{2\mu}\alpha_x, \beta_x(\beta_x)^{2\mu}\big) = (\alpha_x, \beta_x) = (\alpha, \beta)$. Hence, $(\alpha, \beta) \in \Omega_{l+2\mu(l+\kappa)}$. □

Fact 10 *There exist $l \geq 1$ such that $\Omega_l = \Omega_{l+2\mu(l+\kappa)}$.*

Proof. As the constant $(|L|!) \times (|R|!)$ upper bounds the sizes of all sets $\Omega_1, \Omega_2, \ldots$, we know at least one of them is of maximum size. Pick l so that Ω_l is such. Then both $\Omega_l \subseteq \Omega_{l+2\mu(l+\kappa)}$ (by Fact 9) and $|\Omega_l| \geq |\Omega_{l+2\mu(l+\kappa)}|$ (by the selection of l). Necessarily then, the two sets must be equal. □

Intuitively, for the two lengths l and $l + 2\mu(l + \kappa)$, this last fact says that *between two copies of ϑ, every i-I-i path of either length can be replaced by some path of the other length* without M noticing the trick (recall Fact 6).

4. The Proof

We now fix an arbitrary deterministic mole $M = (s, \delta, f)$ over Σ_5 and prove that it fails to solve liveness. To this end, in Sections 4.2 and 4.3 we construct a maze that 'confuses' M. Our most important building blocks are the paths of the next section.

4.1. Three Important Paths

In this section we fix $n := 5$, $i := 2$, $I := \{1, 2\}$. For these n, i, and I, we fix Π, ϑ, κ and μ as in Section 3.4, let λ be a length as in Fact 10, and set $\Lambda := 2\mu(\lambda + \kappa)$.

Lemma 7 *There exist paths $\pi, \varrho, \sigma \in \Pi$ such that*

- *M cannot distinguish among them: $\gamma_\pi = \gamma_\varrho = \gamma_\sigma$.*
- *ϱ is Λ-disjoint on itself, and π is Λ-disjoint on σ.*
- *π is Λ-shorter than ϱ, and ϱ is Λ-shorter than σ: $|\varrho| - |\pi| = |\sigma| - |\varrho| = \Lambda$.*
- *π is non-empty but short: $0 < |\pi| \leq \Lambda$.*

Proof. Each of π, ϱ, σ is a trap on ϑ. We carefully select the infixes $x, y, z \in \Pi$.

We set $\varrho := \vartheta y \vartheta$, where y has length $\lambda + \Lambda$ and guarantees ϱ is Λ-disjoint (cf. Section 2.2, p. 220) on itself. Constructing y is straightforward (Fig. 5a): We pick paths

$$\begin{aligned} \eta &:= \textit{any } 2\text{-}I\text{-}1 \text{ path of length } \lambda, \\ \vartheta' &:= \textit{the } 1\text{-}I\text{-}1 \text{ path of length } \kappa \text{ that is 0-disjoint on } \vartheta, \\ \iota &:= \textit{any } 1\text{-}I\text{-}1 \text{ path of length } \Lambda - (2\kappa + \lambda), \text{ and} \\ \eta' &:= \textit{the } 1\text{-}I\text{-}2 \text{ path of length } \lambda \text{ that is 0-disjoint on } \eta. \end{aligned}$$

Then, setting $y := \eta\vartheta'\iota\vartheta'\eta'$ we see this is indeed a 2-I-2 path of length $\lambda + \Lambda$; and shifting $\varrho = \vartheta y \vartheta = \vartheta\eta\vartheta'\iota\vartheta'\eta'\vartheta$ on a copy of itself by $\Lambda = |\vartheta\eta\vartheta'\iota|$ causes only its prefix $\vartheta\eta\vartheta'$ to overlap with the 'mirroring' suffix $\vartheta'\eta'\vartheta$, so that no vertex is shared (Fig. 5b).

We set $\pi := \vartheta x \vartheta$, where x has length λ and guarantees π is indistinguishable from ϱ. Selecting x is easy: Since y is of length $\lambda + \Lambda$, the inner-behavior (α_y, β_y) of M on ϱ is in $\Omega_{\lambda+\Lambda}$, and therefore in Ω_λ. Hence, there exist λ-long paths that induce this inner-behavior. Picking x to be such, we know $(\alpha_x, \beta_x) = (\alpha_y, \beta_y)$ and hence $\gamma_\pi = \gamma_\varrho$.

We set $\sigma := \vartheta z \vartheta$, where z has length $\lambda + 2\Lambda$ and guarantees that π is Λ-disjoint on σ and that σ is indistinguishable from π. Note that, given the lengths of x and z, the disjointness condition amounts to saying that π and σ should not intersect when 'centered' on top of each other. The construction of z is trickier.

We start by selecting a path y' that is as long as y (namely, of length $\lambda + \Lambda$) and does not intersect with π when the two paths are 'centered' on top of each other (namely, $\vartheta x \vartheta$ is $(\frac{\Lambda}{2} - \kappa)$-disjoint on y'). This selection is trivial: We simply take the unique 1-I-1 path that is as long as π (namely, of length $\lambda + 2\kappa$) and 0-disjoint on it, then extend it by $\frac{\Lambda}{2} - \kappa$ in both directions into any 2-I-2 path.

Now, the inner-behavior $(\alpha_{y'}, \beta_{y'})$ of M on $\vartheta y' \vartheta$ is in $\Omega_{\lambda+\Lambda}$, and hence in Ω_λ. So, we can find a λ-long $x' \in \Pi$ that induces the same behavior, $(\alpha_{x'}, \beta_{x'}) = (\alpha_{y'}, \beta_{y'})$. We set $z := (x')^{(\mu)} \vartheta y' \vartheta (x')^{(\mu-1)} \vartheta x$, the path containing $2\mu + 1$ ϑ-separated paths, all copies of x' except the middle and rightmost ones, which copy y' and x.

The length of z is indeed $\lambda + 2\Lambda$. Moreover, $\sigma = \vartheta z \vartheta$ symmetrically extends y' by $|\vartheta (x')^{(\mu)} \vartheta| = |\vartheta (x')^{(\mu-1)} \vartheta x \vartheta| = \frac{\Lambda}{2} + \kappa$, which in turn symmetrically out-lengths π by $\frac{\Lambda}{2} - \kappa$. Overall, σ symmetrically out-lengths π by Λ without intersecting it. In other words, π is Λ-disjoint on σ. Finally, the inner-behavior (α_z, β_z) of M on σ is

$$\big((\alpha_{x'})^\mu \alpha_{y'} (\alpha_{x'})^{\mu-1} \alpha_x,\ \beta_x (\beta_{x'})^{\mu-1} \beta_{y'} (\beta_{x'})^\mu \big) = \big((\alpha_{x'})^{2\mu} \alpha_x,\ \beta_x (\beta_{x'})^{2\mu} \big) = (\alpha_x, \beta_x),$$

where we used Facts 7 and 8, and the selection of μ. Hence, $\gamma_\sigma = \gamma_\pi$. □

4.2. A Maze of Questions

We start (Fig. 5f) with the two strings $\tau_1 := \texttt{[]}^{3\Lambda} \varrho \texttt{[]}$ and

$$\tau_2 := \texttt{[33]}^{\Lambda-1} \texttt{[32]} \texttt{[22]}^{\Lambda-1} \texttt{[23]} \texttt{[33]}^{\Lambda-1} \texttt{[32,34]} \texttt{[45]} \texttt{[55]}^{\Lambda-1} \texttt{[54]} \texttt{[44]}^{|\pi|-1} \texttt{[23,43]},$$

which are equally long and each is Λ-disjoint on itself (recall the selection of ϱ). Also, in $\tau := \tau_1 \cup \tau_2$ their graphs intersect only at the endpoints of ϱ, so τ is Λ-disjoint on itself, too. This implies τ^i is also Λ-disjoint on itself, for all $i \geq 1$ (Fig. 5g).

Let $T := \{\tau^i \mid i \geq 1\}$ be the set of all powers of τ. Select τ_{L} and τ_{R} as LR- and RL-dilemmas over T. Fix $m = 2^{|Q|} + 1$. The live string $z = \tau_{\text{L}} \tau^m \tau_{\text{R}}$ is also a power of τ and in it we think of the m 'middle' copies of τ as *special*. On this string, we consider the natural maze $\omega = (z, Z) = (z, \{u, v\})$, where $u = (3, 0)$ and $v = (3, |z|)$.

Consider the $|Q|$ computations of the form[6] $\text{COMP}_{M,p,\varepsilon}(\omega)$ that we get as we vary $p \in Q$ and pick $\varepsilon = u$ when p focuses on the left ($\varphi(p) = (\cdot, \texttt{1})$), and $\varepsilon = v$ otherwise. Some of them are infinite (i.e., they loop) or finite but non-crossing (i.e., they hang; or start and end on the same gate). We disregard them and keep only those that are *crossing* (i.e., they start and end in different gates). Let $k \leq |Q|$ be their number.

Fix d to be any of these k computations and fix $1 \leq i \leq m$. We know d 'visits' the ith special copy of τ, and we want to discuss its behavior there. In particular, we want to consider the parity $b_{i,d} \in \{0, 1\}$ of the number of times that d 'fully crosses' the copy of ϱ in the ith special copy of τ. A careful definition of $b_{i,d}$ follows.

If we 'rip off' ϱ from the ith special copy of τ and then add the two endpoints u_i, v_i of the path as new gates, we construct a new maze,

$$\chi_i := \big((\tau_{\text{L}} \tau^{i-1})\ \tau_2\ (\tau^{m-i} \tau_{\text{R}}),\ \ \{u, v, u_i, v_i\} \big).$$

[6] Recall the overloading of operator COMP that we defined in Section 2.3 (p. 221).

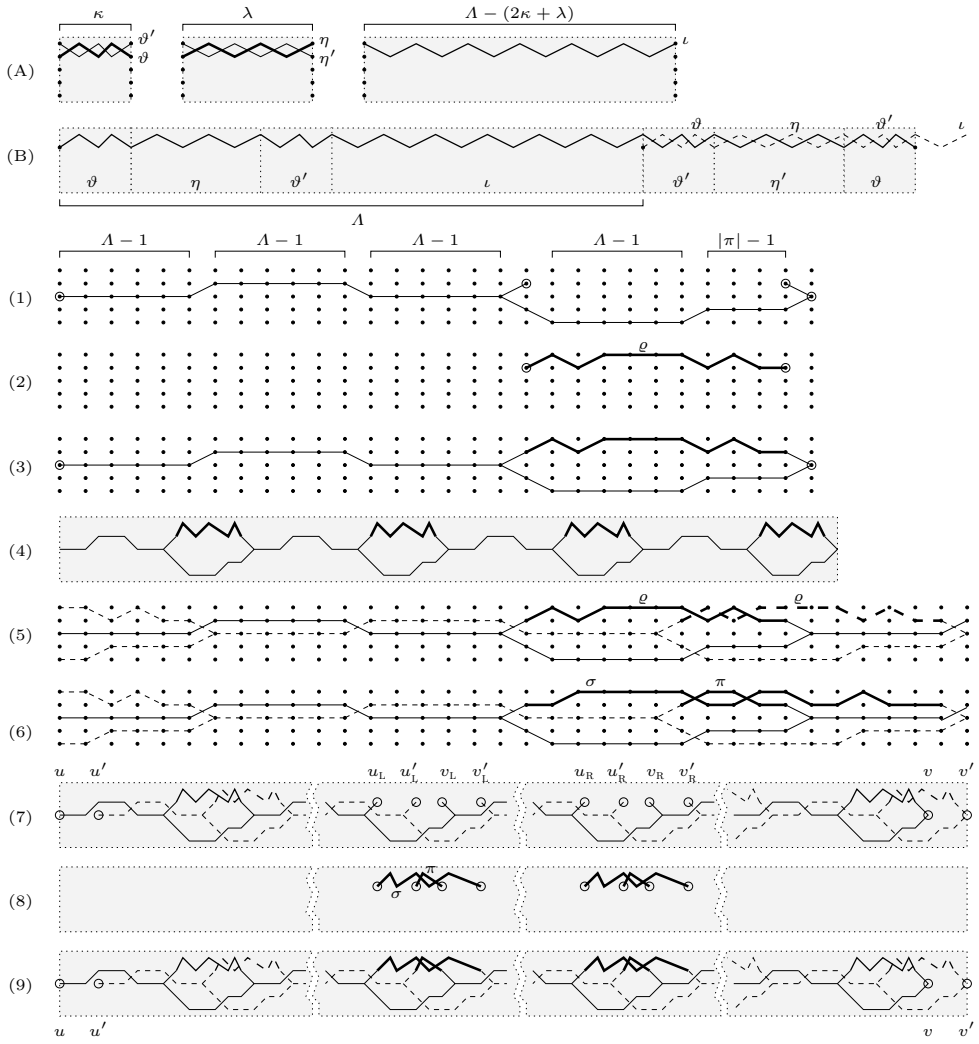


Figure 5: (a) in A: picking η, ι; then the mirrors ϑ', η'. (b) in B: the path ϱ and how it is Λ-disjoint on itself. (c) in each of 1, 2, 3: a 29-long string, 6-disjoint on itself; see 5. (d) in each of 1, 2, 3: a maze; gates marked with circles. (e) in 3: the composition of the mazes of 1, 2. (f) in 1, 2, 3: examples of τ_2, τ_1, τ, respectively, for a schematic case $\Lambda = 6$, $|\pi| = 4$, and a schematic ϱ. (g) in 4: a schematic of τ^4; in 5: a snippet of the union of a τ^i with a Λ-shifted copy of itself. (h) in 7: a schematic of χ', focusing on the snippets around the leftmost, i_1th special, i_2th special, and rightmost pairs of copies of τ. (i) in 8: a schematic of ψ', for the same snippets. (j) in 9: a schematic of $\omega' = \chi' \circ \psi'$, for the same snippets; in 5, 6: a better view of how σ, π connect the two disjoint graphs of x' when they replace two copies of ϱ.

By the 'complementary' operation, where we rip off everything *except* the particular copy of ϱ, we can construct the 'complementary' maze,

$$\psi_i := \big(([]^{|\tau_{\mathrm{L}}\tau^{i-1}|})\ []^{3\Lambda}\varrho[]\ ([]^{|\tau^{m-i}\tau_{\mathrm{R}}|}),\ \ \{u_i, v_i\}\big).$$

Clearly, $\omega = \chi_i \circ \psi_i$, and d is a finite computation on this composition. By Lemma 1, we can break d into its finitely many, finite fragments $d_1, d_2, \ldots, d_\nu$. We know every even(-indexed) fragment is a computation on ψ_i; we call it *crossing* if its starting and ending gates differ. The bit $b_{i,d}$ records the parity of the number of such fragments, namely: $b_{i,d} = 0 \iff d$ exhibits an even number of crossing even fragments.

Intuitively, as the mole develops a crossing computation on ω, each special copy of τ asks: "odd or even?" The mole answers with the parity of the number of times that it fully crosses ϱ in that copy. The bits $b_{i,d}$ record these answers.

Organizing these $m \times k$ bits into m k-long vectors $b_i = (b_{i,d})_d$, for $i = 1, \ldots, m$, we see that there are more vectors than values for them: $2^k \leq 2^{|Q|} < 2^{|Q|} + 1 = m$. Hence, $b_{i_1} = b_{i_2}$ for some $1 \leq i_1 < i_2 \leq m$. Which means that, in each crossing finite computation, the answer to the i_1th question equals the answer to the i_2th one.

4.3. A More Complex Maze

We now return to $\omega = (z, \{u, v\})$. We remove ϱ from the i_1th and i_2th special copies of τ, and name the four natural new gates $u_{\mathrm{L}}, v_{\mathrm{L}}$ (endpoints of ϱ in the i_1th copy) and $u_{\mathrm{R}}, v_{\mathrm{R}}$ (endpoints of ϱ in the i_2th copy) to get the new maze

$$\chi := (x, X) = \big((\tau_{\mathrm{L}}\tau^{i_1-1})\ \tau_2\ (\tau^{i_2-i_1-1})\ \tau_2\ (\tau^{m-i_2}\tau_{\mathrm{R}}),\ \ \{u, v, u_{\mathrm{L}}, v_{\mathrm{L}}, u_{\mathrm{R}}, v_{\mathrm{R}}\}\big).$$

As previously, the 'complementary' maze (remove everything *except* the two ϱ's) is

$$\psi := (y, Y) = \big((\cdots)\ []^{3\Lambda}\varrho[]\ (\cdots)\ []^{3\Lambda}\varrho[]\ (\cdots),\ \ \{u_{\mathrm{L}}, v_{\mathrm{L}}, u_{\mathrm{R}}, v_{\mathrm{R}}\}\big),$$

where ellipses stand for appropriately many []s (namely: $|\tau_{\mathrm{L}}\tau^{i_1-1}|$, $|\tau^{i_2-i_1-1}|$, and $|\tau^{m-i_2}\tau_{\mathrm{R}}|$ []s, respectively). Obviously, $\omega = \chi \circ \psi$. In this section, we will construct a maze $\omega' = \chi' \circ \psi'$, where χ' and ψ' are going to be 'more complex' versions of χ and ψ, respectively.

We start by noting that x is Λ-disjoint on itself (because z is). So, in the union $x' := x \cup ([]^{\Lambda}x)$ of x with a Λ-shifted copy of itself, the two graphs do not intersect. (Fig. 5h.) So, letting $\chi' := (x', X')$, where $X' := X \cup \{u', v', u'_{\mathrm{L}}, v'_{\mathrm{L}}, u'_{\mathrm{R}}, v'_{\mathrm{R}}\}$ contains all gates of χ plus their counterparts in the shifted copy, we know every computation on χ' visits and depends on exactly one of the two disjoint graphs.

Similarly, y is Λ-disjoint on itself (because ϱ is), the union $y \cup ([]^{\Lambda}y)$ contains two pairs of disjoint copies of ϱ, and $Y' := Y \cup \{u'_{\mathrm{L}}, v'_{\mathrm{L}}, u'_{\mathrm{R}}, v'_{\mathrm{R}}\}$ contains their endpoints. Viewing each pair of copies of ϱ as a copy of $\varrho \cup ([]^{\Lambda}\varrho)$, we can replace it with a copy of $\varrho' := \sigma \cup ([]^{\Lambda}\pi)$. If y' is the new sting, we set $\psi' := (y', Y')$. (Fig. 5i.) Crucially, this substitution preserved (i) *the lengths of strings*: $|y'| = |y \cup ([]^{\Lambda}y)|$, because

$$|\varrho'| = |\sigma \cup ([]^{\Lambda}\pi)| = |\sigma| = 2\kappa + \lambda + 2\Lambda = |\varrho| + \Lambda = |\varrho \cup ([]^{\Lambda}\varrho)|;$$

(ii) *the number and disjointness of paths*: since π is Λ-disjoint on σ, we know ϱ' also contains two disjoint paths; and (iii) *the set of endpoints of paths*: e. g., on the copy of

ϱ' on the left, σ and π have endpoints u_{L}, v'_{L} and u'_{L}, v_{L}. Note that every computation on ψ' visits and depends on exactly one of the paths.

Clearly, the graphs of x' and y' intersect only at the gates in Y'. So, χ', ψ' are composable, into $\omega' = (z', Z') := \chi' \circ \psi' = (x' \cup y', \{u, v, u', v'\})$. (Fig. 5j.) Note that u, u' are on the far left; v, v' are on the far right; and the four paths of y' connect the two graphs of x': the mole can switch graphs only if it fully crosses one of the paths.

4.4. The Hidden Gate

Consider the dead input $z'\texttt{[]}$ and the computation $c' = \text{LCOMP}_{M,s}(\vdash z'\texttt{[]}\dashv)$ on it. From now on, our goal is to prove that c' never visits `[]`. Equivalently, that M never visits the 0-degree side of the rightmost node v' of z'. Intuitively, that *the maze implied by z' hides v' from the mole*. This will immediately imply the failure of M: on the *live* input $z'\texttt{[33]}$ the mole will compute exactly as on the *dead* input $z\texttt{[]}$, as it will never visit the 0-degree side of v' to note the difference.

We start by remarking that, since the first symbol of z' is `[33]`, any attempt of the mole to depart from $\vdash$ into a state of focus other than $(3, \texttt{1})$ is followed by a step back to $\vdash$. Ignoring these attempts and also noting that the mole can never move past `[]`, we see that c' consists essentially of zero or more computations of the form $\text{COMP}_{M,p,1}(z'\texttt{[]})$ with $\varphi(p) = (3, \texttt{1})$. For our purposes, it is enough to study the case where c' consists of exactly one such computation.

So, suppose $c' = \text{COMP}_{M,p,1}(z'\texttt{[]})$, where $\varphi(p) = (3, \texttt{1})$. As a mole, every time M visits the 0-degree side of the nodes u', v, v', it changes direction to 'return into the graph' of z'. Call every such move a *turn* and break c' into *segments* $c'_1, c'_2, \dots$ so that successive segments are joined at a turn: the later segment starts at the state and position following the last state and position of the earlier segment. Clearly: each segment is a computation on ω'; $c'_1 = \text{COMP}_{M,p,u}(\omega')$ but later segments start at a gate in $\{u', v, v'\}$; and *either* all segments are finite, in which case their list is finite iff c' is, *or* not, in which case the list is finite and only the last segment is infinite.

To prove that c' never visits `[]`, it is enough to show that no segment ends in v'. This, in turn, is a corollary of the following: (i) c'_1 starts at u, (ii) a finite segment that starts at u and does not hang necessarily ends on either u or v, and (iii) a finite segment that starts at v and does not hang necessarily ends on either u or v. We only show (ii), in the next section. Statement (iii) is similar, and (i) is known.

4.5. The Final Argument

Let d' be a non-hanging finite segment of c' that starts at u. As a finite computation on $\omega' = \chi' \circ \psi'$, it can be broken into finitely many, finite fragments $d'_1, d'_2, \dots, d'_\nu$; odd(-indexed) fragments compute on χ' and even(-indexed) fragments compute on ψ' (Lemma 1). By previous remarks, every odd fragment visits and depends on exactly one of the two graphs (non-shifted and shifted) inside x'; and every even fragment visits and depends on exactly one of the four paths in y'. Calling an even fragment *crossing* if its start and final gates differ, we clearly see that two successive odd fragments visit different graphs in x' iff the even fragment between them is crossing.

Generalizing, and since d' starts on u, *each odd fragment visits the* shifted *graph in* x' *iff the number of crossing even fragments that precede it is* odd.

Towards a contradiction, assume d' does not end in u or v. Then it ends in either u' or v'. Hence, d'_ν is an odd fragment that visits the shifted graph in x'. This implies *the total number of crossing even fragments* (before d'_ν, and so throughout d') *is odd*. In particular, even fragments exist and d'_1 *necessarily ends at a gate in* Y.

To reach a contradiction, we will show that, by replacing every fragment d'_i of d' with an appropriate computation d_i on the original maze ω, we can create a computation d on ω that cannot really exist. Before we start, let $h : X' \to X$ be the mapping of every gate in X' to its unprimed version in X: e. g., $h(u_{\text{L}}) = h(u'_{\text{L}}) = u_{\text{L}}$.

- *If* d'_i *is an odd fragment* (a computation on exactly one of the two graphs in χ') from state q and gate ε to state r and gate ζ, we let d_i be the computation on (the one graph of) χ from q and $h(\varepsilon)$. Clearly, d_i ends at r and $h(\zeta)$. In particular, d_1 starts at $h(u) = u$ and ends at a gate in $h(Y) = Y$.
- *If* d'_i *is an even fragment* (a computation on exactly one of the four paths in ψ') from state q and gate ε to state r and gate ζ, we let d_i be the computation on (one of the two copies of ϱ in) ψ from q and $h(\varepsilon)$. Since ϱ is indistinguishable from each of π and σ, we know d_i ends at r and $h(\zeta)$. Note here the critical use of the inability of the mole to detect the big difference in the lengths of π, ϱ, σ.

Reviewing the list $d_1, d_2, \ldots, d_\nu$, we see that: d_1 starts at $h(u) = u$; for $1 \leq i < \nu$, d_i ends at the state and gate where d_{i+1} starts; d_ν ends on $h(u') = u$ or $h(v') = v$; and every even fragment d_i is crossing (on the path of ψ that it visits) iff d'_i is (on the path of ψ' that it visits). Hence, concatenation builds a computation d on $\chi \circ \psi = \omega$, that starts at u, ends at u or v, and contains an odd number of crossing even fragments.

But is this possible?

If d ends at u, then it never moves beyond τ_{L} (if it did, it would traverse the LR-dilemma and get 'blocked' away from u). In particular, d_1 never reaches a gate in Y. But (by a previous remark) this is where it is supposed to end. Contradiction.

If d ends in v, then it is a crossing computation on ω. As ω equals each of the compositions $\chi \circ \psi$, $\chi_{i_1} \circ \psi_{i_1}$, and $\chi_{i_2} \circ \psi_{i_2}$, we know d can be fragmented in three different ways. Clearly, every *even* fragment with respect to either $\chi_{i_1} \circ \psi_{i_1}$ or $\chi_{i_2} \circ \psi_{i_2}$ is also an *even* fragment with respect to $\chi \circ \psi$, and vice versa; and is *crossing* or not (on the copy of ϱ that it visits) irrespective of which composition we look at it through. So, letting ξ, ξ_1, ξ_2 be the numbers of *crossing even* fragments with respect to the three compositions, we know $\xi = \xi_1 + \xi_2$ and (as established above) ξ is odd. Yet, by the selection of i_1 and i_2, the parities of ξ_1, ξ_2 are respectively $b_{i_1,d}$, $b_{i_2,d}$ and hence equal (as $b_{i_1} = b_{i_2}$), so that ξ should be even. Contradiction. Again. □

5. Conclusion

We discussed the question whether small 2DFAs can solve liveness. We focused on a natural class of restricted but still fully bidirectional 2NFA algorithms, which includes the small 1NFA solvers. We asked whether *small* 2DFAs from that class can succeed and proved that they cannot, *no matter how large they are*.

It is certainly good to provably know that graph exploration alone can never be a sufficient strategy. However, in the context of the full conjecture, the emphasis above stresses an alarming mismatch: a complexity question received a computability answer. This suggests that *the reasons why deterministic moles fail against liveness are only loosely related to the reasons why small* 2DFA*s fail – if they really do.*

In order for our approach in this article to ultimately be of any use against the full conjecture, we need restricted versions of fully bidirectional 2DFAs that are *both* weak enough to succumb to our arguments *and* strong enough to keep us in complexity: large 2DFAs of this kind should be able to solve liveness.

Acknowledgement

More than the usual thanks are due to my PhD research supervisor, Michael Sipser, for his advice and his encouragement throughout this project.

References

[1] B. H. BARNES, A two-way automaton with fewer states than any equivalent one-way automaton. *IEEE Transactions on Computers* **20** (1971) 4, 474–475.

[2] P. BERMAN, A note on sweeping automata. In: *Proceedings of the International Colloquium on Automata, Languages, and Programming*. LNCS 85, Springer, 1980, 91–97.

[3] 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.

[4] J.-C. BIRGET, *Two-way automata and length-preserving homomorphisms*. Report 109, Department of Computer Science, University of Nebraska, 1990.

[5] J.-C. BIRGET, Positional simulation of two-way automata: proof of a conjecture of R. Kannan and generalizations. *Journal of Computer and System Sciences* **45** (1992), 154–179.

[6] M. CHROBAK, Finite automata and unary languages. *Theoretical Computer Science* **47** (1986), 149–158.

[7] V. GEFFERT, C. MEREGHETTI, G. PIGHIZZINI, Converting two-way nondeterministic unary automata into simpler automata. *Theoretical Computer Science* **295** (2003), 189–203.

[8] V. GEFFERT, C. MEREGHETTI, G. PIGHIZZINI, Complementing two-way finite automata. In: *Proceedings of the International Conference on Developments in Language Theory*. LNCS 3572, Springer, 2005, 260–271.

[9] 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*. LNCS 2719, Springer, 2003, 439–451.

[10] R. Kannan, Alternation and the power of nondeterminism. In: *Proceedings of the Symposium on the Theory of Computing*. ACM, 1983, 344–346.

[11] C. Kapoutsis, Removing bidirectionality from nondeterministic finite automata. In: *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*. LNCS 3618, 2005, 544–555.

[12] C. Kapoutsis, *Algorithms and lower bounds in finite automata size complexity.* PhD Thesis, Massachusetts Institute of Technology, June 2006.

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

[14] A. R. Meyer, M. J. Fischer, Economy of description by automata, grammars, and formal systems. In: *Proceedings of the Symposium on Switching and Automata Theory*. IEEE, 1971, 188–191.

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

[16] F. R. Moore, On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on Computers* **20** (1971) 10, 1211–1214.

[17] M. O. Rabin, D. Scott, Finite automata and their decision problems. *IBM Journal of Research and Development* **3** (1959), 114–125.

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

[19] J. I. Seiferas, Untitled manuscript. Communicated to Michael Sipser, Oct. 1973.

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

(Received: October 26, 2005; revised: December 5, 2006)