Journal of Automata, Languages and Combinatorics **u** (v) w, x–y

MINICOMPLEXITY [1]

Christos A. Kapoutsis [2]
LIAFA, Université Paris VII, France
e-mail: christos.kapoutsis@liafa.univ-paris-diderot.fr

ABSTRACT

This is an overview of *minicomplexity*, i.e., of the *complexity of two-way finite automata*. We start with a smooth introduction to its basic concepts, which also brings together several seemingly detached old theorems. We then record some recent advances, both in the theory itself and in its relation to the space complexity of Turing machines. Our exposition follows, extends, and advocates the classic framework of Sakoda and Sipser.

Keywords: Two-way finite automata, descriptional complexity, state complexity, minicomplexity, Turing machines, space complexity

1. Introduction

In Theory of Computation, the distinction between *computability* and *complexity* is clear. In computability, we ask whether a given problem can be solved by a Turing machine (TM), namely whether the problem is *computable*. In complexity, we focus exclusively on problems that indeed can be solved, and we ask how much of a TM's resources they require, the main resources of interest being *time* and *space*.

This distinction is also valid for finite automata (FAs). In FA-*computability*, we ask whether a problem can be solved by a FA; often, but not always, this is the same as asking whether the problem is *regular*. In FA-*complexity*, we focus exclusively on problems that indeed can be solved, and we ask how much of a FA's resources they require; often, but not always, the resource of interest is *size* (as expressed, e.g., by the number of states). Hence, much like the theory of TMs, the theory of FAs also consists of a computability and a complexity component.

This distinction is not widely realized. Specifically, the complexity component is often overlooked. Standard textbooks essentially identify the entire theory of FAs with FA-computability (e.g., see [30, Chapter 1]), barely addressing any FA-complexity issues (e.g., as in [30, Problems 1.60-1, 1.65]). Perhaps one might try to justify this systematic neglect by claiming that these issues are not really a theory; they are just a

[1] Version of a lecture delivered at the 14th International Workshop on Descriptional Complexity of Formal Systems (Braga, Portugal, July 23–25, 2012).

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

list of detached observations on the relative succinctness of FAs. We disagree. Before explaining why, let us discuss another systematic neglect.

This is the systematic neglect of *two-way* FAs (2FAs, whose input head can move in both directions) in favor of *one-way* FAs (1FAs, whose input head can move only forward). Standard textbooks essentially identify FAs with 1FAs (e.g., see [20, Chapters 3–16]), only briefly addressing 2FAs, if at all, as a natural generalization (e.g., as in [20, Chapters 17–18]). As before, one might perhaps try to justify this systematic neglect by pointing out that 2FAs are no more powerful than 1FAs [27], and are thus worthy of no special attention. We again disagree.

Once we realize that the theory of FAs is neither only about computability nor only about one-way automata, we are rewarded with the meaningful, elegant, and rich *complexity theory of two-way finite automata*: a mathematical theory with all standard features of a complexity theory, including computational problems, complexity classes, reductions, and completeness; with challenging, decades-old open questions; and with strong links to TM-complexity and logic. Unfortunately, this theory has eluded the systematic attention of researchers for a long time now. One goal of the present article is to help repair this ... public-relations disaster.

In the title, we already make a solid first step. We suggest for this theory a catchy (hopefully) new name. We call it *minicomplexity*, because we view it as a 'miniature version' of the standard complexity theory of TMs.

We then present the theory in two steps. First (Section 2), we present the fragment which concerns 1FAs, focusing on determinism, nondeterminism, and alternation. By a series of examples of computational problems of increasing difficulty, we introduce complexity classes, reductions, and completeness, also discussing the differences from the respective concepts of TM-complexity. All problems and proofs are elementary; our main goal is to show how a list of old, seemingly detached facts about the relative succinctness of 1FAs are really part of one coherent complexity theory. Then, in our second step (Section 3), we move on to 2FAs. We focus on the Sakoda-Sipser conjecture and on two stronger variants of it, recording their history and some recent advances. Subsequently, we discuss alternation and the relationship between minicomplexity and the space complexity of TMs of at most logarithmic space. We conclude in Section 4.

1.1. Notation

For $h \geq 0$, we let $[h]$ be $\{0, \ldots, h-1\}$, and let $[\![h]\!]$ be the powerset of $[h]$. Our 2FAs are tuples $(S, \Sigma, \delta, q_{\mathrm{s}}, q_{\mathrm{a}})$ of a state set, an alphabet, a transition function, a start state, and an accept state. Our *finite transducers* (2DFTs, 1DFTs) are as in [19].

A (*promise*) *problem* over Σ is a pair $\mathfrak{L} = (L, \tilde{L})$ of disjoint subsets of Σ^*. Every $w \in L \cup \tilde{L}$ is an *instance* of $\mathfrak{L}$, and is *positive*, if $w \in L$, or *negative*, if $w \in \tilde{L}$. To solve $\mathfrak{L}$ is to accept all $w \in L$ but no $w \in \tilde{L}$. The *reverse*, *complement*, *conjunctive star*, and *disjunctive star* of $\mathfrak{L}$ are respectively the problems $\mathfrak{L}^{\mathrm{R}} := (L^{\mathrm{R}}, \tilde{L}^{\mathrm{R}})$, $\neg\mathfrak{L} := (\tilde{L}, L)$,

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

where $\#x_1\#\cdots\#x_l\#$ means that $l \geq 0$, each $x_i \in L \cup \tilde{L}$, and $\#$ is a fresh symbol. Easily,

$$\neg(\bigwedge \mathfrak{L}) = \bigvee \neg\mathfrak{L} \qquad \neg(\mathfrak{L}^{\mathrm{R}}) = (\neg\mathfrak{L})^{\mathrm{R}} \qquad \left(\bigwedge \mathfrak{L}\right)^{\mathrm{R}} = \bigwedge \mathfrak{L}^{\mathrm{R}}$$
$$\neg(\bigvee \mathfrak{L}) = \bigwedge \neg\mathfrak{L} \qquad\qquad\qquad \left(\bigvee \mathfrak{L}\right)^{\mathrm{R}} = \bigvee \mathfrak{L}^{\mathrm{R}}$$

by the definitions. The *conjunctive concatenation* $\mathfrak{L} \wedge \mathfrak{L}'$ of two problems $\mathfrak{L}, \mathfrak{L}'$ is defined in [16]. Families of promise problems admit analogous operations. For more careful definitions, see [16].

2. One-Way Automata

2.1. Size Complexity Basics

Let $h \geq 1$, and consider the following elementary computational problem:

$\vdash \boxed{i}\boxed{\alpha} \dashv$ *Given a number $i \in [h]$ and a set $\alpha \subseteq [h]$, check that $i \in \alpha$.*

The input tape is shown on the left. Every instance fits in just two tape cells, because we use the large alphabet $\Sigma := [h] \cup [\![h]\!]$. The instance is surrounded by the two endmarkers $\vdash$ and $\dashv$, a feature unimportant for 1FAs but essential for 2FAs. Moreover, every instance is promised to be of this form, namely a number followed by a set; all other strings over Σ are irrelevant to this computational problem.

Solving this problem is trivial. We easily design a 1DFA $M = ([h], \Sigma, .\,, 0, 0)$ whose transition function implements the following obvious algorithm:

$\vdash \boxed{3}\boxed{\substack{0\\4}\,1} \dashv$ $0 \to 0 \to 3$ $\quad \vdash \boxed{3}\boxed{\substack{0\\4}\,3} \dashv$ $0 \to 0 \to 3 \to 0 \to 0$

From state 0 on $\vdash$, move to state 0 on the 1st cell. Reading i, move to state i on the 2nd cell. Reading α in state i, check whether $i \in \alpha$. If not, then hang. Otherwise, move to state 0 on $\dashv$. Then fall off $\dashv$, again in state 0.

Two computations, on a negative and a positive instance (for $h = 5$), are shown on the left. Note how M is designed only for instances of the promised form.

Now consider the reverse of this problem, where the set precedes the number:

$\vdash \boxed{\alpha}\boxed{i} \dashv$ *Given a set $\alpha \subseteq [h]$ and a number $i \in [h]$, check that $i \in \alpha$.*

This problem is again trivial. We easily design a 1DFA $M^{\mathrm{R}} = ([\![h]\!], \Sigma, .\,, \emptyset, \emptyset)$:

$\vdash \boxed{\substack{0\\4}\,1}\boxed{3} \dashv$ $\emptyset \to \emptyset \to \substack{0\\4}\,1$ $\quad \vdash \boxed{\substack{0\\4}\,3}\boxed{3} \dashv$ $\emptyset \to \emptyset \to \substack{0\\4}\,3 \to \emptyset \to \emptyset$

From state $\emptyset$ on $\vdash$, move to state $\emptyset$ on the 1st cell. Reading α, move to state α on the 2nd cell. Reading i in state α, check whether $i \in \alpha$. If not, then hang. Otherwise, move to state $\emptyset$ on $\dashv$. Then fall off $\dashv$, again in state $\emptyset$.

However, M^{R} uses 2^h states, whereas M uses only h states. Moreover, this is not due to poor design; we can easily see that M^{R} could not have done significantly better:

> *Proof.* Suppose there is a 1DFA solver X with $< 2^h - 1$ states. For each $\emptyset \neq \alpha \subseteq [h]$, the prefix $\vdash\alpha$ forces X to cross its right boundary (or else X hangs earlier, and fails to accept $\vdash\alpha i\dashv$, for $i \in \alpha$); let q_α be the state after this crossing. Since the states of X are fewer than the non-empty subsets of $[h]$, there exist distinct $\emptyset \neq \alpha, \beta \subseteq [h]$ with $q_\alpha = q_\beta$. If $i \in (\alpha \setminus \beta) \cup (\beta \setminus \alpha)$, then X treats αi and βi identically, a contradiction.

Therefore, the reverse problem is indeed substantially different from the original.

To capture this difference formally, we first introduce a meaningful name for the original problem: we call it MEMBERSHIP. We then note that this "problem" is in fact a *family of problems*, consisting of one different member for every $h \geq 1$:

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

In turn, each family member is a *promise problem* over the alphabet $\Sigma_h := [h] \cup [\![h]\!]$:

$$\text{MEMBERSHIP}_h := (\ \{i\alpha \mid \alpha \subseteq [h]\ \ \&\ \ i \in \alpha\},\ \{i\alpha \mid \alpha \subseteq [h]\ \ \&\ \ i \in \overline{\alpha}\}\ )\,. \tag{1}$$

At the same time, our "algorithm" for MEMBERSHIP has been a *family of* 1DFAs:

$$\mathcal{M} := (M_h)_{h\geq 1} \qquad \text{with}\ \ M_h := ([h], \Sigma_h, .\,, 0, 0)\,.$$

Likewise, the reverse "problem" is a family $\text{MEMBERSHIP}^{\text{R}} := (\text{MEMBERSHIP}^{\text{R}}_h)_{h\geq 1}$, with the h-th member as in (1) but with the order of i and α reversed; and the corresponding "algorithm" is a 1DFA family $\mathcal{M}^{\text{R}} := (M^{\text{R}}_h)_{h\geq 1}$, with $M^{\text{R}}_h := ([\![h]\!], \Sigma_h, .\,, \emptyset, \emptyset)$.

(Hence, all this time we have been using the terms "problem"/"algorithm" and a single description in terms of h to informally refer to and describe an entire family of promise problems/FAs. This is standard practice, which we will repeat. Also, instead of the traditional n, which is reserved for denoting input length, the name for the important parameter is h, for "$h$eight" and because h looks like n.)

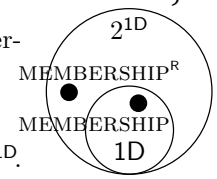
In our new formal terms, the substantial difference between MEMBERSHIP and $\text{MEMBERSHIP}^{\text{R}}$ is that the former can be solved by a family of 1DFAs which grow linearly with h (each M_h has h states), whereas the latter can be solved by some family of exponential growth (each M^{R}_h has 2^h states) and by no family of sub-exponential growth. To state this more succinctly, we introduce the complexity classes $\mathsf{1D}$ and 2^{1D} of all problems which admit 1DFA algorithms with at most polynomially or at most exponentially many states, respectively. More carefully,

$$\mathsf{1D} := \left\{ (\mathfrak{L}_h)_{h\geq 1} \;\middle|\; \begin{array}{l} \text{for some polynomial } p \text{ and 1DFA family } (M_h)_{h\geq 1}, \\ \text{every } M_h \text{ solves } \mathfrak{L}_h \text{ with } \leq p(h) \text{ states} \end{array} \right\}, \tag{2}$$

and similarly for 2^{1D}, with $2^{p(h)}$ instead of $p(h)$. Then our observations so far are summarized by the two statements

$$\text{MEMBERSHIP} \in \mathsf{1D} \quad \text{and} \quad \text{MEMBERSHIP}^{\text{R}} \in 2^{\mathsf{1D}} \setminus \mathsf{1D}\,,$$

and by the map on the side, including the obvious fact $\mathsf{1D} \subseteq 2^{\mathsf{1D}}$.

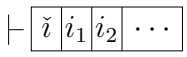


From now on, we will informally say that an algorithm or automaton which is described in terms of h is "small" if the h-th member of the implicit family of finite automata has $\leq p(h)$ states, for some polynomial p and all h.

2.2. More Problems

The profile of MEMBERSHIP is shared by several other elementary problems that have appeared sporadically in the literature. Below, we list some of these problems. When appropriate, the endnotes explain who introduced them and why.(1)

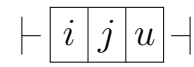
We start with a variant of MEMBERSHIP, where the set is replaced by a list. We call it ∃EQUALITY. The alphabet consists of $[h]$ and $\{\check{\imath} \mid i \in [h]\}$, a tagged copy of $[h]$ which is used for distinguishing the query number:



Given a tagged $i \in [h]$ and a list $i_1, i_2, \ldots, i_l \in [h]$, check that $i = i_j$ for some j.

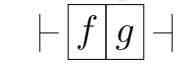
Easily, ∃EQUALITY $\in \mathsf{1D}$ but ∃EQUALITY$^{\mathsf{R}} \in 2^{\mathsf{1D}} \setminus \mathsf{1D}$. The same holds for a variant problem, which we call SORTED ∃EQUALITY, where the numbers in the list are promised to be strictly increasing: $i_1 < i_2 < \cdots < i_l$.(2)

The next two problems are called PROJECTION and COMPOSITION.(3) (4) The first one uses the alphabet $[h] \cup [h]^h$ of all numbers in $[h]$ and all h-tuples over $[h]$:



Given a number $i \in [h]$, an index $j \in [h]$, and a tuple $u \in [h]^h$, check that $i = u(j)$.

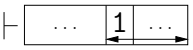
The second problem uses the alphabet $([h] \rightharpoonup [h])$ of all partial functions on $[h]$:



Given two functions $f, g : [h] \rightharpoonup [h]$, check that $f(g(0)) = 0$.

Easily, PROJECTION and COMPOSITION$^{\mathsf{R}}$ are in $\mathsf{1D}$, but their reverses are in $2^{\mathsf{1D}} \setminus \mathsf{1D}$.

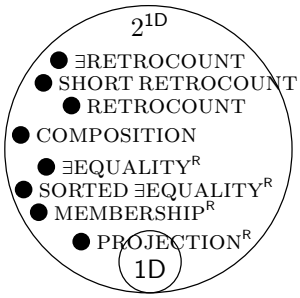
Finally, a classic. We call it RETROCOUNT, for the obvious reason:(5)



Given a binary string, check that its h-th rightmost bit is 1.

Easily, RETROCOUNT$^{\mathsf{R}} \in \mathsf{1D}$ but RETROCOUNT $\in 2^{\mathsf{1D}} \setminus \mathsf{1D}$. The same is true for two variant problems: ∃RETROCOUNT, where we must check that a 1 exists at some distance from the end which is a multiple of h; and SHORT RETROCOUNT, where the input is promised to be of length $< 2h$.(6)

The map on the right summarizes our observations so far. All problems shown in it are in $2^{\mathsf{1D}} \setminus \mathsf{1D}$, whereas all their reverses are in $\mathsf{1D}$. Perhaps this looks like an unstructured list of detached observations on how reversal affects the size of 1DFAs. We will soon see that this map does contain some structure. Even at this early level, we can classify problems in terms of hardness, and use the resulting lattice to deduce algorithms and lower bounds.

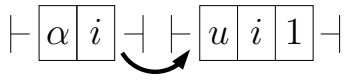


2.3. Reductions

Roughly speaking, a problem reduces to another 'in one-way polynomial size' if a *small* 1DFT can convert its positive/negative instances to positive/negative instances of the other problem, respectively, with only a *small* increase in height.

For example, consider the following simple algorithm for converting instances of MEMBERSHIP$^{\mathsf{R}}_h$ to instances of PROJECTION$^{\mathsf{R}}_h$:

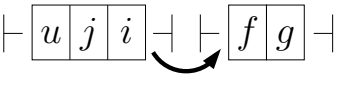
Reading $\alpha \subseteq [h]$, print the characteristic vector of α, namely the tuple $u \in [2]^h$ with $u(x) = 1$ iff $x \in \alpha$, for all x. Reading $i \in [h]$, print the two-symbol string $i1$.



Clearly, every instance αi of MEMBERSHIP$^{\mathsf{R}}_h$ is mapped into a string $ui1$ which is indeed an instance of PROJECTION$^{\mathsf{R}}_h$; and αi is positive iff $i \in \alpha$, namely iff $u(i) = 1$, namely iff $ui1$ is also positive. Moreover, this conversion can be easily implemented by a 1-state 1DFT and involves no increase of h across the two problems.

For another example, consider converting PROJECTION$^{\mathsf{R}}_h$ to COMPOSITION$_{h^2}$ by the following algorithm, which can be implemented by an $(h{+}1)$-state 1DFT:

Reading $u \in [h]^h$; print a characteristic function of u, $f : [h^2] \to [2]$ such that $f(y{\cdot}h{+}x) = 0$ iff $u(y) = x$, for all x, y. Reading $j \in [h]$, store j. Reading i, print any function $g : [h^2] \rightharpoonup [h^2]$ such that $g(0) = j{\cdot}h{+}i$.



Easily, every instance uji maps to an instance fg which satisfies $f(g(0)) = 0$ iff $u(j) = i$. Moreover, the increase in height, from h to h^2, is small.

Formally, for two families of promise problems $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1}$ and $\mathcal{L}' = (\mathfrak{L}'_h)_{h\geq 1}$, we write $\mathcal{L} \leq_{\mathsf{1D}} \mathcal{L}'$ and say $\mathcal{L}$ *reduces to* $\mathcal{L}'$ *in one-way polynomial size*, if there exist a family of 1DFTs $(T_h)_{h\geq 1}$ and two polynomial functions s and e such that every T_h has $s(h)$ states and maps instances of $\mathfrak{L}_h$ to instances of $\mathfrak{L}'_{e(h)}$ so that

$$x \in L_h \implies T_h(x) \in L'_{e(h)} \qquad \text{and} \qquad x \in \tilde{L}_h \implies T_h(x) \in \tilde{L}'_{e(h)}\,,$$

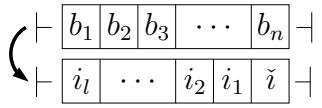
for all x. In the special case where $s(h) = 1$ for all h, every T_h is nothing more than a mapping from symbols to strings. We then say that $\mathcal{L}$ *homomorphically reduces to* $\mathcal{L}'$ and write $\mathcal{L} \leq_{\mathrm{h}} \mathcal{L}'$. Hence, by our two algorithms above, we have already shown:

$$\textsc{membership}^{\mathsf{R}} \leq_{\mathrm{h}} \textsc{projection}^{\mathsf{R}} \leq_{\mathsf{1D}} \textsc{composition}\,,$$

with $s(h) = 1$, $e(h) = h$ and with $s(h) = h{+}1$, $e(h) = h^2$, respectively.

A final, more interesting example, which involves problems of arbitrarily long instances, is the reduction of $\exists\textsc{retrocount}_h$ to $\exists\textsc{equality}^{\mathsf{R}}_h$:

Scan the input bits $b_1 b_2 \cdots b_n$; whenever $b_j = \mathtt{0}$, print nothing; whenever $b_j = \mathtt{1}$, print j mod h untagged. On reaching $\dashv$, print $(n{+}1)$ mod h tagged.



To see why this works, note that the input instance is positive iff it has a $\mathtt{1}$ among all b_j which satisfy $j = n{-}\lambda h{+}1$ for some $\lambda \geq 1$, namely $j = (n{+}1)$ mod h. Equivalently, the instance is positive iff the critical value $(n{+}1)$ mod h appears in the list of the modulo-h values of all positions of $\mathtt{1}$s. So, the algorithm outputs this list, followed by the critical value. Easily, this can be implemented by an h-state 1DFT which simply keeps track of the index of the current position modulo h. Therefore $\exists\textsc{retrocount} \leq_{\mathsf{1D}} \exists\textsc{equality}^{\mathsf{R}}$, with $s(h) = e(h) = h$.

One-way polynomial-size reductions do have the two nice properties we would expect from them. The first one is, of course, transitivity:

$$\mathcal{L} \leq_{\mathsf{1D}} \mathcal{L}' \quad \& \quad \mathcal{L}' \leq_{\mathsf{1D}} \mathcal{L}'' \implies \mathcal{L} \leq_{\mathsf{1D}} \mathcal{L}''\,. \tag{3}$$

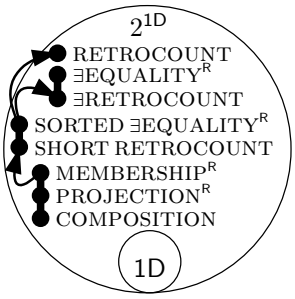
This holds because we can combine an s_1-state 1DFT T_1 with an s_2-state 1DFT T_2 in standard cartesian-product style to build an $s_1{\cdot}s_2$-state 1DFT which outputs $T_2(T_1(x))$ for all x. Notice that, if s_1, e_1 and s_2, e_2 are the polynomial functions associated with the assumption of (3), then the corresponding functions for the conclusion are $s_1(h){\cdot}s_2(e_1(h))$ and $e_2(e_1(h))$; hence, homomorphic reductions are also transitive.

The second nice property of one-way polynomial-size reductions is that our complexity classes are closed under them. For example,

$$\mathcal{L} \leq_{\mathsf{1D}} \mathcal{L}' \quad \& \quad \mathcal{L}' \in \mathsf{1D} \implies \mathcal{L} \in \mathsf{1D}\,, \tag{4}$$

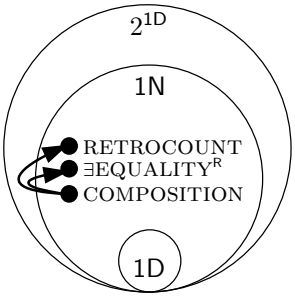
and similarly for 2^{1D}. This time, from an s_1-state 1DFT T and an s_2-state 1DFA M we get an $s_1{\cdot}s_2$-state 1DFA which accepts x iff M accepts $T(x)$, for all x. If the polynomial

functions associated with the assumption of (4) are s_1, e_1 and s_2, then the respective function for the conlusion is $s_1(h)\cdot s_2(e_1(h))$.



By transitivity and a few more reductions, we can arrive at the lattice on the right, where arrows and lines denote $\leq_{1D}$ in one and in both directions, respectively. So, using the closures and this lattice, we can now derive all observations of Section 2.2 from just the three facts that MEMBERSHIP$^R \notin$ 1D and that ($\exists$) RETROCOUNT $\in 2^{1D}$.

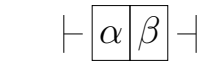
An analogy is now clear, between 1D, 2^{1D}, and $\leq_{1D}$ on one hand, and well-known settings of TM-complexity on the other: P, EXP, and polynomial-time reductions; or L, PSPACE, and logarithmic-space reductions. Therefore, a natural next question is whether our map also contains an analog of NP and NL.



2.4. Nondeterminism

The class 1N is defined as in (2), but for 1NFAs. In order to place it on our map, we note that 1D $\subseteq$ 1N $\subseteq 2^{1D}$ (by the definitions and by the subset construction [23]), that RETROCOUNT, $\exists$EQUALITY$^R \in$ 1N (easily [21]), and that 1N is closed under $\leq_{1D}$ (by adapting the argument for (4)). The result is shown on the side.

A natural next step is to look for problems in $2^{1D} \setminus$ 1N. We thus consider the following problem, defined over $[\![h]\!]$, which we call INCLUSION:



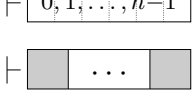
Given two sets $\alpha, \beta \subseteq [h]$, check that $\alpha \subseteq \beta$.

We also consider two related problems: SET EQUALITY, which asks whether $\alpha = \beta$; and DISJOINTNESS, which asks whether $\alpha \cap \beta = \emptyset$.(7) Clearly, all three problems are in 2^{1D}. However, none is in 1N. The argument for INCLUSION is a classic:

> *Proof.* Suppose there is a 1NFA solver X with $< 2^h$ states. Let $\alpha_0, \ldots, \alpha_{2^h-1}$ list all subsets of $[h]$ so that the characteristic vector of each α_i is the binary representation of i. Consider the $2^h \times 2^h$ matrix with (i,j)-th entry the instance $\alpha_i\alpha_j$. Clearly, every $\alpha_i\alpha_i$ on the diagonal is positive, so X accepts it; pick an accepting branch, and let q_i be its state right after crossing into the second set. Since the states $q_0, \ldots, q_{2^h-1}$ outnumber those of X, we have $q_i = q_j$ for some $i > j$. By a standard cut-and-paste argument, this implies that X accepts $\alpha_i\alpha_j$. Since $i > j$, some 1 in the representation of i is a 0 for j, so $\alpha_i \not\subseteq \alpha_j$. Therefore, X accepts a negative instance, a contradiction.

The argument for SET EQUALITY is identical. As for DISJOINTNESS, the simple 1DFT which replaces β with $\overline{\beta}$ proves that INCLUSION $\leq_{1D}$ DISJOINTNESS (easily); hence, by the closure of 1N under $\leq_{1D}$, this problem is also not in 1N.

Two more problems in this category are ROLL CALL and EQUAL ENDS:(8) (9)



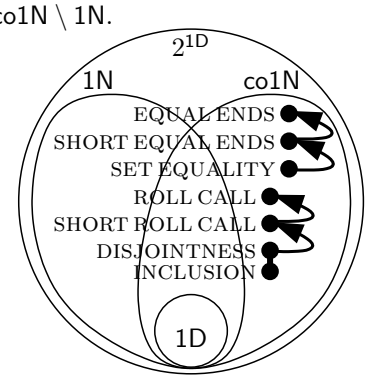
Given a list of numbers from $[h]$,
check that every number appears at least once.

Given a binary string,
check that its h-long prefix and suffix are equal.

We also consider their restrictions SHORT ROLL CALL and SHORT EQUAL ENDS, where every instance is promised to be of length $\leq 2h$.[10] Easily, all four problems are in 2^{1D}. But none of them is in $\mathsf{1N}$, because DISJOINTNESS $\leq_{\mathrm{h}}$ SHORT ROLL CALL and SET EQUALITY $\leq_{\mathrm{h}}$ SHORT EQUAL ENDS, by straightforward reductions.

Before we update our map with the new problems, we note one more feature that they have in common: although they do not admit small 1NFAs, their complements do. E.g., $\neg$INCLUSION$_h$ is solved by an $(h{+}1)$-state 1NFA which 'guesses' an $i \in \alpha \setminus \beta$. This leads us to the class $\mathsf{co1N}$, which is defined as in (2) but with "M_h solves $\neg\mathfrak{L}_h$". Hence, all seven of our new problems are actually in $\mathsf{co1N} \setminus \mathsf{1N}$.

The new map is on the right. We have used a few easy reductions, together with the chain $\mathsf{1D} \subseteq \mathsf{co1N} \subseteq 2^{\mathsf{1D}}$ (since $\mathsf{1D} \subseteq \mathsf{1N} \subseteq 2^{\mathsf{1D}}$ and because $\mathsf{1D}$ and 2^{1D} are closed under complement). Of course, all problems shown here have their complements in $\mathsf{1N} \setminus \mathsf{co1N}$. Also, every problem from Section 2.2 is in $\mathsf{1N} \cap \mathsf{co1N}$: this holds, e.g., because ($\exists$) RETROCOUNT and its complement homomorphically reduce to each other (by simply flipping the bits) and because $\mathsf{co1N}$ is closed under $\leq_{\mathsf{1D}}$ (since $\mathsf{1N}$ is).



Now, two natural questions are whether $\mathsf{1N}$ also contains complete problems, by analogy to NP and NL; and whether we can also find problems in $2^{\mathsf{1D}} \setminus (\mathsf{1N} \cup \mathsf{co1N})$. The answers are postponed for Section 2.6 and Section 2.7. In the meantime, the next section examines $\mathsf{1N} \cap \mathsf{co1N}$ a bit more carefully.

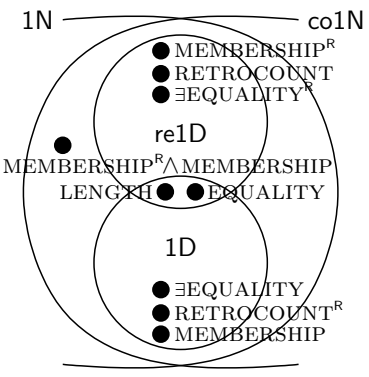
2.5. Complement and Reversal

In Section 2.1 we saw that $\mathsf{1D}$ is not closed under reversal, and in Section 2.4 we saw that $\mathsf{1N}$ is not closed under complement. In contrast, $\mathsf{1D}$ is closed under complement (every 1DFA can be 'complemented' by swapping accepting and rejecting states, after adding a fresh 'sink' state [23]) and $\mathsf{1N}$ is closed under reversal (every 1NFA can be 'reversed' by reversing all transitions, before adding a fresh start state [23]). Hence,

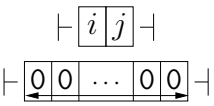
$$\mathsf{co1D} = \mathsf{1D} \neq \mathsf{re1D} \qquad \text{and} \qquad \mathsf{co1N} \neq \mathsf{1N} = \mathsf{re1N}\,,$$

where $\mathsf{co1D}$ is defined as $\mathsf{co1N}$ but for 1DFAs, and the classes $\mathsf{re1D}$ and $\mathsf{re1N}$ are defined as in (2) but with "M_h solves $\mathfrak{L}_h^{\mathrm{R}}$".

To place the new class $\mathsf{re1D}$ on our map, we note that it is in $\mathsf{1N} \cap \mathsf{co1N}$ (by $\mathsf{1D} \subseteq \mathsf{1N}$ and the two closures which we just mentioned), and that both the difference $(\mathsf{1N} \cap \mathsf{co1N}) \setminus (\mathsf{1D} \cup \mathsf{re1D})$ and the intersection $\mathsf{1D} \cap \mathsf{re1D}$ are non-empty. For the difference, consider the conjunctive concatenation (cf. Section 1) of any two problems which witness the two sides of the symmetric difference of $\mathsf{1D}$ and $\mathsf{re1D}$, e.g., MEMBERSHIP$^{\mathrm{R}}$ and MEMBERSHIP: this is unsolvable by small 1DFAs



in either direction (as each of the original problems $\leq_{\mathrm{h}}$-reduces to it), but small 1NFAs can solve both it and its complement (easily). As for $\mathsf{1D} \cap \mathsf{re1D}$, this contains any elementary problem which is the reverse of itself, for example:



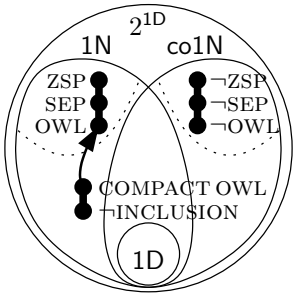
Given two numbers $i, j \in [h]$, check that $i = j$.

Given a string $w \in \{0\}^*$, check that $|w| = h$.

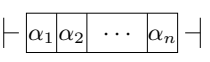
We call these two problems EQUALITY and LENGTH, respectively.(11)

2.6. Completeness

A problem is 1N-*complete* if it is in 1N and every other problem in 1N reduces to it. This notion was introduced by Sakoda and Sipser [24] together with the first proof of 1N-completeness, for a problem which we call ONE-WAY LIVENESS (or OWL).(12) Earlier, Seiferas [26] had introduced two problems which would also turn out to be 1N-complete; we call them SEPARABILITY (or SEP) and ZERO-SAFE PROGRAMS (or ZSP).(13) (14)

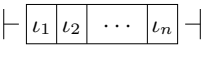


For SEPARABILITY, the alphabet is $[\![h]\!]$. The definition uses the notion of a 'block': this is any string of sets $\alpha_0\alpha_1\cdots\alpha_l$ in which the first one contains the number of those that follow, namely $\alpha_0 \ni l$. E.g., {0,2,4}∅{1,4} and {0,2} are blocks, but ∅ and {0,2,4}{1,4} are not blocks. Then, a list of sets is 'separable' if it can be split into blocks.



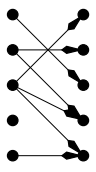
Given a list of sets $\alpha_1, \alpha_2, \ldots, \alpha_n \subseteq [h]$, check that it is separable.

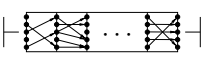
For ZERO-SAFE PROGRAMS, the alphabet is $[h] \cup [h]^2$. A symbol ι is seen as an instruction, applied to a variable $\xi \subseteq [h]$: an $i \in [h]$ means "remove i from ξ"; a $(j, i) \in [h]^2$ means "if $j \in \xi$, then add i to ξ". A string $\iota_1\iota_2\cdots\iota_n$ is seen as a program. We call it '0-safe' if, for every initial value of ξ, it satisfies: $0 \in \xi$ at start $\Longrightarrow 0 \in \xi$ in the end.



Given a program $\iota_1, \iota_2, \ldots, \iota_n \in [h] \cup [h]^2$, check that it is 0-safe.

For ONE-WAY LIVENESS, the alphabet is $\mathbb{P}([h]^2)$. A symbol γ is seen as a 2-column graph of height h, with its pairs $(i, j) \in \gamma$ as rightward arrows. A string $\gamma_1\gamma_2\cdots\gamma_n$ is seen as an h-tall, $(n{+}1)$-column graph. We call this graph 'live' if column $n{+}1$ is reachable from column 1.





Given a h-tall, one-way, multi-column graph, check that it is live.

We also call COMPACT OWL the restriction where all instances have length $n = 2$.

The completeness of OWL (under $\leq_{\mathrm{h}}$) was proved in [24]. The completeness of SEP was announced there, too (see [13], for an outline of OWL $\leq_{\mathrm{h}}$ SEP). Here, we show the completeness of ZSP, which seems not to have been announced before.

Proof. We copy the intuition of [26, p. 3] to homomorphically reduce SEP_h to ZSP_{2h}.

Before we start, let us consider how a 2^h-state 1DFA solves SEP_h. The strategy is to keep track of the 'overhang set' of the input so far. This is the set ξ of all $i \in [h]$ such that, if the input has exactly i more symbols, then it will definitely be separable. At start, $\xi = \{0\}$, because the empty string is separable and more symbols may render it non-separable. From then on, each time we read an $\alpha \subseteq [h]$, we update ξ as follows:

(i) Every non-zero $i \in \xi$ is replaced by $i{-}1$: Since the input before α would definitely be separable with i more symbols, it follows that the current input up to α will definitely be separable with $i{-}1$ more symbols.

(ii) If $0 \in \xi$, then 0 is replaced by all $i \in \alpha$: Since the input before α was separable, it follows that the current input up to α will definitely be separable if α starts a new block, consisting of it and i more symbols, for some $i \in \alpha$.

In the end, we accept iff the input is separable with no more symbols, namely iff $0 \in \xi$.

We are now ready to describe a 1-state 1DFT reducing SEP_h to ZSP_{2h}. To map an instance $x = \alpha_1\alpha_2\cdots\alpha_n$ of SEP_h to an instance $y = p_1p_2\cdots p_n$ of ZSP_{2h}, the machine keeps track of the overhang set ξ of x using the set variable of y. Specifically, each p_k is a sub-program which applies to ξ the updates (i)-(ii) caused by the set α_k.

For (i), we just list all instruction pairs of the form $(i, i{-}1)\, i$, for $i \neq 0$:

$$(1,0)1 \quad (2,1)2 \quad \cdots \quad (h{-}1, h{-}2)h{-}1\,. \tag{+}$$

Easily, each pair replaces i by $i{-}1$, if $i \in \xi$, or does nothing, if $i \notin \xi$. And we list them by increasing i, so that newly-added values do not interfere with pre-existing ones. As for (ii), we may just use the instructions $(0, i)$ for all $i \in \alpha_k$, plus a final instruction 0:

$$\big((0,i)\big)_{i\in\alpha_k} 0\,. \tag{×}$$

Easily, this replaces 0 with all $i \in \alpha_k$, if $0 \in \xi$, or does nothing, if $0 \notin \xi$.

However, there is a problem: (×) must precede (+), because all tests for $0 \in \xi$ must precede $(1, 0)$, which may add 0 to ξ; at the same time, (+) must precede (×), because every test for an element $i \neq 0$ must precede $(0, i)$, which may add i to ξ.

This is why we reduce to ZSP_{2h}, and not just to ZSP_h: so that ξ has both a 'lower half' in $[h]$ and an 'upper half' in $[2h] \setminus [h]$. This way, we can implement (ii) in two stages, one before and one after (+). The first stage uses one instruction $(0, h{+}i)$ for every $i \in \alpha_k$ to copy α_k into the upper half of ξ, 'above' all pre-existing values, followed by the instruction 0 to remove 0. Then, after (+) has correctly updated the lower half, the second stage uses one instruction pair $(h{+}i, i)h{+}i$ for each $i \in \alpha_k$ to transpose the copy of α_k from the upper to the lower half, leaving the upper half empty again.

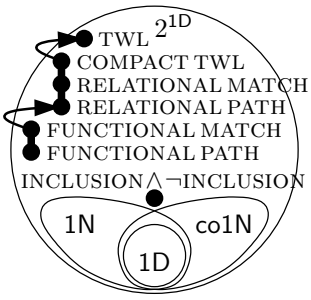
Overall, our 1DFT replaces every $\alpha_k \subseteq [h]$ in its input x with the sub-program p_k:

$$\big((0,h{+}i)\big)_{i\in\alpha_k} 0 \qquad (1,0)1\ (2,1)2\ \cdots\ (h{-}1,h{-}2)h{-}1 \qquad \big((h{+}i,i)h{+}i\big)_{i\in\alpha_k}\,.$$

The final result is program y. By our discussion above, x is separable iff its overhang set contains 0 in the end, which holds iff y transforms $\{0\}$ into a superset of $\{0\}$. In turn, this holds iff y is 0-safe (for the 'only if' direction, note that if we start with a superset of $\{0\}$ then the intermediate sets cannot possibly shrink).

Finally, it is interesting (and easy) to note that COMPACT OWL is $\leq_{\text{1D}}$-equivalent to the complement of DISJOINTNESS, and thus also to the complement of INCLUSION.

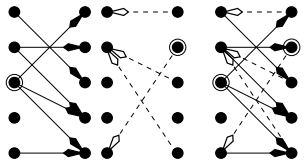
2.7. Harder Problems



For a problem in $2^{1D} \setminus (1N \cup co1N)$, we may consider the conjunctive concatenation of any two problems from the two sides of the symmetric difference of 1N and co1N, e.g., INCLUSION $\wedge$ ¬INCLUSION: this is $\leq_h$-harder than both INCLUSION and ¬INCLUSION, but still in 2^{1D}. Two more interesting examples are RELATIONAL MATCH and RELATIONAL PATH.[15] [16]

For RELATIONAL MATCH, the alphabet is two copies of $\mathbb{P}([h]^2)$, one of them tagged. Every symbol $A \subseteq [h]^2$ is seen as an h-tall, 2-column graph with rightward arrows (as in OWL), if it is untagged, or with leftward arrows, if tagged. A pair $A\check{B}$ is seen as the overlay (union) of the corresponding two graphs. If this overlay contains a cycle, we say that the binary relations A and B 'match'.

$\vdash \boxed{A}\boxed{\check{B}} \dashv$ *Given a relation $A \subseteq [h]^2$ and a tagged $B \subseteq [h]^2$, check that A and B match.*



For RELATIONAL PATH, the alphabet includes in addition two copies of $[h]$, one of which is tagged. A string $iA\check{B}\check{\jmath}$ is seen again as the overlay for $A\check{B}$, except this time the i-th left-column node and the j-th right-column node are marked respectively as 'entry' and 'exit' points.

$\vdash \boxed{i}\boxed{A}\boxed{\check{B}}\boxed{\check{\jmath}} \dashv$ *Given $i \in [h]$ and $A \subseteq [h]^2$, and tagged $B \subseteq [h]^2$ and $j \in [h]$, check that the resulting overlay has an entry-to-exit path.*

Finally, we call FUNCTIONAL MATCH and FUNCTIONAL PATH the respective restrictions where both relations are promised to be partial functions, i.e., in $([h] \rightharpoonup [h])$.[17] [18]

To place these problems on the map, note that FUNCTIONAL MATCH $\notin$ co1N (since ¬DISJOINTNESS $\leq_{1D}$-reduces to it, by a 1DFT which maps $\alpha\beta$ to $A\check{B}$ where $A := \{(i,i) \mid i \in \alpha\}$ and $B := \{(i,i) \mid i \in \beta\}$) and that FUNCTIONAL PATH $\notin$ 1N [9]. Therefore, both problems are outside $1N \cup co1N$, provided that they reduce to each other. Indeed they do, but to prove so we need two new, variant concepts.

First, we need the variant problem FUNCTIONAL ZERO-MATCH. This asks only that A and B '0-*match*', namely that their overlay contains a cycle through the 0-th left-column node.[19] This is known to be $\leq_h$-equivalent to FUNCTIONAL MATCH [19].

Second, we need a variant '*nondeterministic* one-way polynomial-size reduction', denoted by $\leq_{1N}$. This differs from $\leq_{1D}$ in that the underlying transducer T is nondeterministic (a 1NFT) and such that every input x causes all accepting branches to produce the same output $T(x)$. Easily, $\leq_{1N}$ is also transitive and 1N is closed under it.

We now prove that FUNCTIONAL MATCH and FUNCTIONAL PATH are $\leq_{1N}$-equivalent. First, we replace the first problem by its equivalent FUNCTIONAL ZERO-MATCH. Then, we reduce FUNCTIONAL ZERO-MATCH$_h$ to FUNCTIONAL PATH$_{h+1}$ by a 1-state 1DFT which adds a fresh node h serving both as entry and as exit, directs the entry h into the path out of 0, and redirects 0 to the exit h:

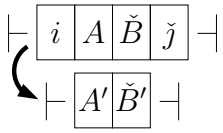
Reading A, print hA', where $A'(t) := A(t)$ for $t \neq 0, h$, whereas $A'(h) := A(0)$ and $A'(0) := h$. Reading $\check{B}$, print $\check{B}'\check{h}$, where $B'(t) := B(t)$ for $t \neq h$, whereas $B'(h)$ is undefined.

$\vdash \boxed{A}\boxed{\check{B}} \dashv$

$\vdash \boxed{h}\boxed{A'}\boxed{\check{B}'}\boxed{\check{h}} \dashv$

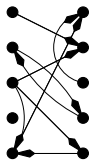
Conversely, we reduce FUNCTIONAL PATH$_h$ to FUNCTIONAL ZERO-MATCH$_{h+1}$ by the simple h-state 1NFT which first transposes $A, \check{B}$ from $[h]$ to $[h{+}1] \setminus \{0\}$, then connects the 0-th left-column node to the transposed entry (via the 0-th right-column node), and the transposed exit back to the 0-th left-column node:

Reading i, store i. Reading A, print A', where $A'(0) := 0$ and $A'(t) := A(t{-}1){+}1$ for $t \neq 0$. Reading $\check{B}$ and recalling i, guess $j' \in [h]$; print $\check{B}'$, where $B'(0) := i{+}1$, $B'(j'{+}1) := 0$, and $B'(t) := B(t{-}1){+}1$ for $t \neq 0, j'{+}1$; and store j' in place of i. Reading j and recalling j', accept iff $j = j'$.



Note how nondeterminism allows the machine to use the exit j before reading it.

Appropriately adjusted, both of the above reductions work even when A and B are relations. Hence, RELATIONAL MATCH and RELATIONAL PATH are also $\leq_{\mathsf{1N}}$-equivalent, through the variant RELATIONAL ZERO-MATCH of the first problem, which is again known to be $\leq_{\mathrm{h}}$-equivalent to it [19].(20)



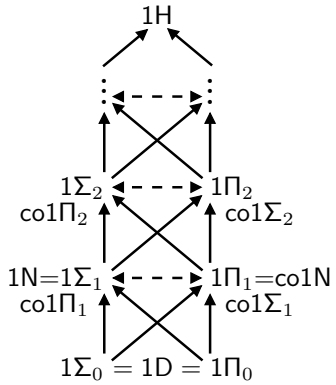
Finally, we also introduce TWO-WAY LIVENESS (or TWL).(21) This generalizes OWL to the alphabet $\mathbb{P}([2h]^2)$ of all h-tall, 2-column graphs with arbitrary arrows between the vertices. Its restriction to instances of length 2 is called COMPACT TWL and is $\leq_{\mathsf{1D}}$-equivalent to RELATIONAL MATCH (by a modification of [19, Lemma 8.1]).(22) Hence, COMPACT TWL and TWL are also outside $\mathsf{1N} \cup \mathsf{co1N}$ —as well as (easily) inside 2^{1D}.

2.8. Alternation

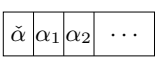
The classes $\mathsf{1N}$ and $\mathsf{co1N}$ are on the first level of a *one-way polynomial-size hierarchy*, defined by analogy to the polynomial-time hierarchy and to the space alternating hierarchies of TM-complexity. For each $k \geq 1$, we define the class $\mathsf{1\Sigma}_k$ (resp., $\mathsf{1\Pi}_k$) as in (2) but for 1Σ_kFAs (1Π_kFAs), namely for alternating 1FAs performing $< k$ alternations between existential and universal states, starting with existential (universal) ones. We also let $\mathsf{1\Sigma}_0, \mathsf{1\Pi}_0 := \mathsf{1D}$ and $\mathsf{1H} := \bigcup_{k \geq 0} (\mathsf{1\Sigma}_k \cup \mathsf{1\Pi}_k)$. Then

$$\mathsf{1D} \;\subseteq\; \mathsf{1\Sigma}_k, \mathsf{1\Pi}_k \;\subseteq\; \mathsf{1\Sigma}_{k+1}, \mathsf{1\Pi}_{k+1} \;\subseteq\; \mathsf{1H}$$

(by the definitions) and $\mathsf{co1\Sigma}_k = \mathsf{1\Pi}_k$ and $\mathsf{1\Sigma}_k = \mathsf{co1\Pi}_k$ (easily), for all $k \geq 0$, causing $\mathsf{co1H} = \mathsf{1H}$.



The natural question is whether this hierarchy of classes is strict. For the first level, we already know (Section 2.4), for example, that ¬INCLUSION $\in \mathsf{1\Sigma}_1 \setminus \mathsf{1\Pi}_1$ and that INCLUSION $\in \mathsf{1\Pi}_1 \setminus \mathsf{1\Sigma}_1$. For the higher levels, we use the (much more powerful) theorems of [4]. We start with a 'core' problem, which we call ∃INCLUSION:(23)



Given a tagged $\alpha \subseteq [h]$ and a list $\alpha_1, \alpha_2, \ldots, \alpha_l \subseteq [h]$, check that $\alpha \subseteq \alpha_j$ for some j.

Then, by alternate applications of conjunctive and disjunctive star (cf. Section 1), we build the following table of witnesses for all levels, where 'INCL' abbreviates

'INCLUSION' (for $k = 1$, reversal is redundant but we use it, for symmetry):

$1\Sigma_1\backslash 1\Pi_1$	$1\Sigma_2\backslash 1\Pi_2$	$1\Sigma_3\backslash 1\Pi_3$	$1\Sigma_4\backslash 1\Pi_4$	$1\Sigma_5\backslash 1\Pi_5$	$\cdots$
$\neg\text{INCL}^{\text{R}}$	$\exists\text{INCL}^{\text{R}}$	$\bigvee\neg\exists\text{INCL}^{\text{R}}$	$\bigvee\bigwedge\ \exists\text{INCL}^{\text{R}}$	$\bigvee\bigwedge\bigvee\neg\exists\text{INCL}^{\text{R}}$	$\cdots$
INCL^{R}	$\neg\exists\text{INCL}^{\text{R}}$	$\bigwedge\ \exists\text{INCL}^{\text{R}}$	$\bigwedge\bigvee\neg\exists\text{INCL}^{\text{R}}$	$\bigwedge\bigvee\bigwedge\ \exists\text{INCL}^{\text{R}}$	$\cdots$
$1\Pi_1\backslash 1\Sigma_1$	$1\Pi_2\backslash 1\Sigma_2$	$1\Pi_3\backslash 1\Sigma_3$	$1\Pi_4\backslash 1\Sigma_4$	$1\Pi_5\backslash 1\Sigma_5$	$\cdots$

(5)

In fact, [4] shows that all lower bounds in this table for $k \geq 2$ are valid even for 2FAs (see also the discussion in Section 3.4).

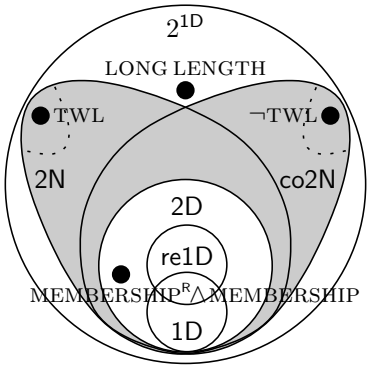
3. Two-Way Automata

3.1. The Sakoda-Sipser Conjecture

For 2FAs, the main complexity classes, analogous to 1D and 1N, are 2D and 2N. To place them on the map, we observe that $\mathsf{1D} \subseteq \mathsf{2D} \subseteq \mathsf{2N} \subseteq 2^{\mathsf{1D}}$ (by the definitions and [27]), that $\mathsf{2D} = \mathsf{re2D}$ and $\mathsf{2N} = \mathsf{re2N}$ (easily), and that $\mathsf{2D} = \mathsf{co2D}$ (by [28]). Therefore, the chain

$$\mathsf{1D}, \mathsf{re1D} \subsetneq \mathsf{2D} \subseteq \mathsf{2N}, \mathsf{co2N} \subsetneq 2^{\mathsf{1D}}$$

mentions all interesting classes. The first inclusion is strict, witnessed by, e.g., $\text{MEMBERSHIP}^{\text{R}} \wedge$ MEMBERSHIP (cf. Section 2.5). The last inclusion is also strict, because of the variant of LENGTH (cf. Section 2.5) for length 2^h, which we call LONG LENGTH (by [3, Fact 5.2], and then by [7, Cor. 4.3]). The *Sakoda-Sipser conjecture* [24] says that the middle inclusion is also strict: $\mathsf{2D} \neq \mathsf{2N}$. In fact, it is believed that even $\mathsf{2N} \neq \mathsf{co2N}$ [7].



A two-way head is very powerful. All problems mentioned in Sections 2.1–2.7 are in 2N. In fact, all of them are known to be already in 2D, except for:

- ONE-WAY LIVENESS, SEPARABILITY, and ZERO-SAFE PROGRAMS;
- COMPACT TWL, RELATIONAL MATCH, and RELATIONAL PATH; and
- TWO-WAY LIVENESS.

The first three problems are $\leq_{\text{h}}$-equivalent (Section 2.6) and thus 2D contains either all three or none of them (as it is closed under $\leq_{\text{h}}$ [24, 14]). The next three problems reduce to each other under $\leq_{\mathsf{1D}}$ or $\leq_{\mathsf{1N}}$ (Section 2.7). However, all six of the reductions among them can be easily replaced by $\leq^{\text{lac}}_{\mathsf{2D}}$-reductions, namely '(*two-way*) *polynomial-size/print reductions*' [19]. These differ from $\leq_{\mathsf{1D}}$ in that the underlying transducer is two-way (a 2DFT) and restricted to print only poly(h) times on its output tape. So, 2D again contains either all three or none of the problems in the second group (as 2D is also closed under $\leq^{\text{lac}}_{\mathsf{2D}}$ [19]).

Overall, we are essentially left with only three problems that may witness the Sakoda-Sipser conjecture: TWL, and its two severe restrictions to one-way graphs and to two-symbol graphs, namely to OWL and to COMPACT TWL, respectively.

The full problem is actually 2N-complete under $\leq_{\mathrm{h}}$ [24]. Hence, by the closure of 2D and 2N under $\leq_{\mathrm{h}}$ [24], we get the following equivalences:

$$\mathsf{2D} = \mathsf{2N} \iff \textsc{twl} \in \mathsf{2D} \qquad \text{and} \qquad \mathsf{2N} = \mathsf{co2N} \iff \neg\textsc{twl} \in \mathsf{2N}\,.$$

Therefore, the Sakoda-Sipser conjecture is concretely reformulated as $\textsc{twl} \notin \mathsf{2D}$, i.e.:

> *no* poly(h)*-state* 2DFA *can check that an* h*-tall, two-way, multi-column graph contains a path from its leftmost to its rightmost column.* (6)

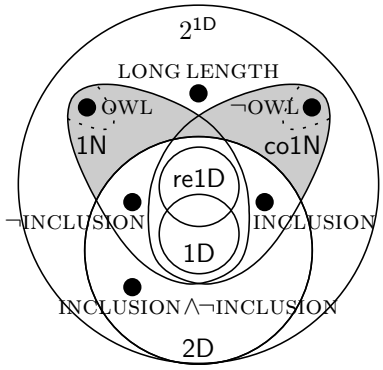
Similarly, the concrete reformulation of the conjecture $\mathsf{2N} \neq \mathsf{co2N}$ is that $\neg\textsc{twl} \notin \mathsf{2N}$, namely that:

> *no* poly(h)*-state* 2NFA *can check that an* h*-tall, two-way, multi-column graph contains no path from its leftmost to its rightmost column.* (7)

In the next two sections we discuss stronger versions of these two conjectures.

3.2. A Stronger Conjecture I

The restriction of TWL to OWL leads to a stronger conjecture, also from [24], that even $\textsc{owl} \notin \mathsf{2D}$, namely that (6) holds even if the graph is one-way. Given that OWL is 1N-complete, this is equivalent to $\mathsf{1N} \not\subseteq \mathsf{2D}$, which was conjectured already in [26]. This stronger claim has been the focus of most attacks against the Sakoda-Sipser conjecture over the past four decades. The typical strategy has been to confirm the claim for some subclass of 2DFAs of restricted bidirectionality or restricted information, as in the following list of results.



- In [26], a fairly simple argument confirmed the claim for *single-pass* 2DFAs, namely for 2DFAs which halt upon reaching an end-marker (cf. Note 13). However, this provably avoided the full claim, because, as noted also in [26], unrestricted small 2DFAs are strictly more powerful (cf. Note 9).
- In [29], a breakthrough argument confirmed the claim for *sweeping* 2DFAs, namely for 2DFAs which reverse their head only on end-markers. The structure and tools of that proof have since been copied and reused several times (e.g., see [12, 16, 15]). Again, the result provably avoided the full claim, because unrestricted small 2DFAs are strictly more powerful [29, by J. Seiferas].
- In [8], a reduction argument confirmed the claim for *almost oblivious* 2DFAs, namely for 2DFAs whose number of distinct trajectories over n-long inputs is only $o(n)$ (instead of the $2^{O(n \log n)}$ maximum). The proof first showed that small 2DFAs of this kind are as powerful as sweeping ones, then made black-box use of [29]. As a result, the full claim was once again provably avoided.
- In [12], a computability argument confirmed the claim for 2DFA *moles*, namely for 2DFAs which explore the multi-column graph of an instance of OWL as a 'system of tunnels'. In fact, the proof showed that $\textsc{owl}_5$ is already unsolvable by 2DFA moles of any size. Hence, it also completely avoided the full claim.

- In [15], a recent argument confirmed the claim for 2DFAs *with few reversals*, namely for 2DFAs whose number of head reversals is only $o(n)$ (instead of the $O(n)$ maximum). This again avoided the full claim, because, as shown also in [15], unrestricted small 2DFAs are strictly more powerful. In fact, more recent arguments show that few-reversal 2DFAs necessarily perform only $O(1)$ reversals [18, Theorem 1] and that an infinite hierarchy of computational power exists below them [18, Theorem 2].

In conclusion, if the full claim OWL $\notin$ 2D is false, then this can only be due to a multi-pass 2DFA algorithm which uses the full information of every symbol and, infinitely often, performs $\Theta(n)$ reversals and exhibits $\Omega(n)$ trajectories.

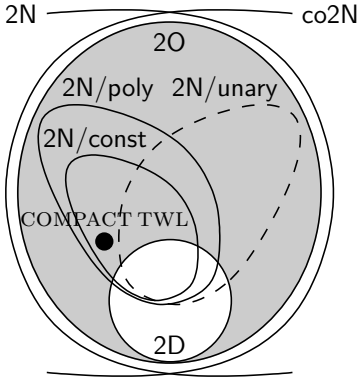
For the second conjecture of Section 3.1, a similar strengthening is also possible: we believe that even $\neg$OWL $\notin$ 2N, namely that (7) holds even if the graph is one-way. One advance in this direction concerns 2NFAs of restricted bidirectionality:

- In [11], it was confirmed that $\neg$OWL admits no small *sweeping* 2NFAs.

A tractable next goal could be to confirm the same for 2NFAs with few reversals.

3.3. A Stronger Conjecture II

The restriction of TWL to COMPACT TWL leads to the stronger conjecture, which was suggested in [19], that even COMPACT TWL $\notin$ 2D, i.e., that (6) holds even for three-column graphs. This is part of an approach in which we focus on subclasses of 2N for restricted instance length, and ask whether 2D contains any of them. Specifically, we introduce the subclasses



$$\mathsf{2N/const} \subsetneq \mathsf{2N/poly} \subsetneq \mathsf{2N/exp} \subsetneq \mathsf{2N}$$

of problems whose instances are of length constant, polynomial, or exponential in h, respectively. For example, COMPACT TWL $\in$ 2N/const and SORTED $\exists$EQUALITY $\in$ 2N/poly, since both problems are in 2N and their instances are of length ≤ 2 and $\leq h{+}1$, respectively; but LENGTH $\notin$ 2N/exp, since this problem has (negative) instances of arbitrary length. Our conjecture says that, not only are all three of these subclasses not in 2D, but even COMPACT TWL, a problem with the shortest interesting instance length, is not in 2D. To better understand the meaning of this conjecture, two remarks are due.

First, COMPACT TWL does admit sub-exponential 2DFAs. This can be shown directly, by applying Savitch's algorithm [25]. However, it also follows from a more general phenomenon, which involves *outer-nondeterministic* 2FAs (2OFAs, namely 2NFAs which perform nondeterminstic choices only on the end-markers), and the respective class 2O. We know that 2N/poly $\subseteq$ 2O (a simple argument [19]) and that 2O $\subseteq$ 2DSIZE($2^{\mathrm{poly}(\log h)}$) (a theorem of [5], which uses [25]). Hence, COMPACT TWL admits quasi-polynomially large 2DFAs because all problems in 2O do.

Let us also note that, like $\mathsf{2N/poly}$, the class $\mathsf{2N/unary}$ of all unary problems of $\mathsf{2N}$ is also in $\mathsf{2O}$ (by [6]), and that $\mathsf{2D}$ contains either subclass iff it contains the entire $\mathsf{2O}$:

$$\mathsf{2N/poly} \subseteq \mathsf{2D} \iff \mathsf{2O} \subseteq \mathsf{2D} \iff \mathsf{2N/unary} \subseteq \mathsf{2D} \tag{8}$$

(because each subclass shares with $\mathsf{2O}$ a common complete problem, for appropriate reductions under which $\mathsf{2D}$ is closed [19]). Moreover, $\mathsf{2O} = \mathsf{co2O}$ (another theorem of [5]), so that this entire discussion takes place inside $\mathsf{2N} \cap \mathsf{co2N}$.

The second remark about our stronger conjecture is the equivalence [19]:

$$\text{COMPACT TWL} \notin \mathsf{2D} \iff \text{RELATIONAL MATCH} \not\leq_{\mathrm{h}} \text{FUNCTIONAL MATCH}\,. \tag{9}$$

This connects our conjecture on the left-hand side, which is clearly an *algorithmic* statement, to the purely *combinatorial* statement on the right-hand side:

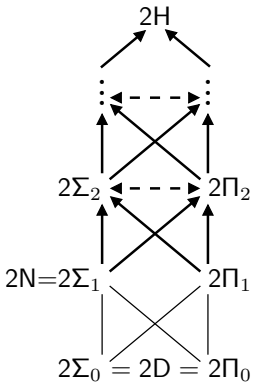
> *no pair of systematic ways of replacing h-tall relations by* poly(h)*-tall functions respects the existence of cycles.* (10)

Therefore, in regard to the well-known dichotomy of intuition betweeen algorithms and combinatorics for upper and lower bounds respectively, we see that both sides are supported: *to disprove the conjecture*, one may focus on the left-hand side of (9) and search for a small algorithm for liveness on h-tall, two-way, two-symbol graphs; *to prove the conjecture*, one may focus on the right-hand side and search for a proof of (10). We continue this discussion in Section 3.5.

3.4. Alternation Again

The *two-way polynomial-size hierarchy* is defined as in Section 2.8 but for 2FAs. By the same witnesses as for 1FAs, we know this hierarchy does not collapse either [4]. Yet, there are two important differences. First, the witnesses of (5) work only for $k \geq 2$; for $k = 1$, we still do not know:

- whether the inclusion $\mathsf{2\Sigma}_0 \subseteq \mathsf{2\Sigma}_1$ is strict,
- whether the inclusion $\mathsf{2\Pi}_0 \subseteq \mathsf{2\Pi}_1$ is strict, and
- whether $\mathsf{2\Sigma}_1$ and $\mathsf{2\Pi}_1$ are incomparable.



Second, although the first of these questions is indeed the Sakoda-Sipser conjecture, the last two are *not* about $\mathsf{2D}$ vs. $\mathsf{co2N}$ and $\mathsf{2N}$ vs. $\mathsf{co2N}$: for $k \geq 1$, it is open whether $\mathsf{co2\Sigma}_k = \mathsf{2\Pi}_k$ (and thus also whether $\mathsf{2\Sigma}_k = \mathsf{co2\Pi}_k$). In fact, Geffert [4, §7] conjectures that $\mathsf{co2\Pi}_1$ is not even in $\mathsf{2H}$, and thus that $\mathsf{co2H} \neq \mathsf{2H}$.

3.5. Relation to Turing Machine Complexity

Minicomplexity is related to logarithmic-space TM-complexity as shown below. The main link is that logarithmic-space deterministic TMs with short advice can simulate logarithmic-space nondeterministic TMs ($\mathsf{L/poly} \supseteq \mathsf{NL}$) iff small 2DFAs can simulate small 2NFAs on short inputs ($\mathsf{2D} \supseteq \mathsf{2N/poly}$). This remains true if we reduce space to $\log\log n$ and advice to poly($\log n$) ($\mathsf{LL/polylog} \supseteq \mathsf{NLL}$) and lengthen the inputs to

$2^{\mathrm{poly}(h)}$ ($\mathsf{2D} \supseteq \mathsf{2N/exp}$). The problems SHORT TWL and LONG TWL are the restrictions of TWL to inputs of promised length $\leq h$ and $\leq 2^h$, respectively.(24)

$$\begin{array}{ccccccc}
 & & & & \mathsf{2D} \supseteq \mathsf{2N/const} & \Longrightarrow & \mathsf{2D} \ni \text{COMPACT TWL} \\
 & & & & \Uparrow & & \Uparrow \\
\mathsf{L} \supseteq \mathsf{NL} & \Longrightarrow & \mathsf{L/poly} \supseteq \mathsf{NL} & \overset{[2,\,14]}{\Longleftrightarrow} & \mathsf{2D} \supseteq \mathsf{2N/poly} & \overset{[14,\,19]}{\Longleftrightarrow} & \mathsf{2D} \ni \text{SHORT TWL} \\
\Uparrow{\scriptstyle [31]} & & & & \Uparrow & & \Uparrow \\
\mathsf{LL} \supseteq \mathsf{NLL} & \Longrightarrow & \mathsf{LL/polylog} \supseteq \mathsf{NLL} & \overset{[14]}{\Longleftrightarrow} & \mathsf{2D} \supseteq \mathsf{2N/exp} & \overset{[14]}{\Longleftrightarrow} & \mathsf{2D} \ni \text{LONG TWL} \\
 & & & & \Uparrow & & \Uparrow \\
 & & & & \mathsf{2D} \supseteq \mathsf{2N} & \overset{[24]}{\Longleftrightarrow} & \mathsf{2D} \ni \text{TWL}
\end{array}$$

Note that, by (8), $\mathsf{2D} \supseteq \mathsf{2N/unary}$ is yet another reformulation of $\mathsf{L/poly} \supseteq \mathsf{NL}$. Also note that all inclusions and memberships in this diagram are conjectured to be false.

This diagram can be interpreted in two ways. On one hand, people interested in space-bounded TMs can see that $\mathsf{2D}$ vs. $\mathsf{2N}$ offers a unifying setting for studying L vs. NL. Confirming $\mathsf{2D} \not\supseteq \mathsf{2N}$ could be a first step in a gradual attack towards $\mathsf{2D} \not\supseteq \mathsf{2N/exp}$ and $\mathsf{2D} \not\supseteq \mathsf{2N/poly}$, hence towards $\mathsf{LL} \neq \mathsf{NLL}$ and $\mathsf{L} \neq \mathsf{NL}$. Or, confirming $\mathsf{2D} \not\ni$ COMPACT TWL (or its combinatorial version (10)) could be a single step towards $\mathsf{L} \neq \mathsf{NL}$. On the other hand, people stydying $\mathsf{2D}$ vs. $\mathsf{2N}$ can use this diagram to appreciate the difficulty of proving a separation on bounded-length inputs.

Analogous diagrams can be drawn for other modes. For example, replacing $\mathsf{2N}$ by its counterpart $\mathsf{2A}$ for alternating 2FAs, we get $\mathsf{L/poly} \supseteq \mathsf{P} \iff \mathsf{2D} \supseteq \mathsf{2A/poly}$ [14], since alternating logarithmic-space coincides with deterministic polynomial time.

4. Conclusion

This was an overview of the *complexity of two-way finite automata*, or *minicomplexity*. We presented it in the classic Sakoda-Sipser framework, to emphasize the tight analogy with the standard theory of the complexity of Turing machines. We believe this view helps reveal important structure. It is, of course, a coarse view, unable to distinguish beyond polynomial differences. For finer views, at the level of asymptotic or exact values, one resorts to the more standard vocabulary in terms of 'trade-offs'.

We focused on one-way and two-way heads and on the deterministic, nondeterministic, and alternating modes. However, minicomplexity also involves other heads (rotating, sweeping) and other standard modes from TM-complexity (probabilistic, quantum, interactive); see [13] for a broader view. It can also mimic TM-complexity in other ways. For example, see [17] for a first step towards *descriptive minicomplexity*, where minicomplexity classes receive logical characterizations.

The selection of the presented material reflects the author's immediate interests, as well as space restrictions. The effort to systematically record, organize, and present material of this kind continues online, at `www.minicomplexity.org`.

Acknowledgment

Many thanks to Viliam Geffert for his kind help with some of his theorems from [4].

Notes

(1) MEMBERSHIP: Introduced in [16, p. 459], as a problem whose reverse can be used as 'core' for building witnesses of separations of classes. See also [15, Eq. (7)].

(2) SORTED ∃EQUALITY: Introduced in [21, Prop. 3] (essentially), as a problem whose reverse has logarithmically-small 1-pebble 2DFAs, but admits no small 1DFAs.

(3) PROJECTION: Introduced in [21, Prop. 2] (essentially), as a problem whose reverse witnesses the asymptotic value of the trade-off in converting 2DFAs to 1DFAs.

(4) COMPOSITION: Introduced in [22, p. 1213] (essentially), as a problem which witnesses the asymptotic value of the trade-off in converting 2DFAs to 1DFAs.

(5) RETROCOUNT: Introduced in [21] (attributed to M. Paterson), as a simple problem which witnesses the asymptotic value of the trade-off in converting 1NFAs to 1DFAs.

(6) SHORT RETROCOUNT: Introduced in [21] (essentially), for the purpose of restricting RETROCOUNT to finitely many instances which still admit no small 1DFAs.

(7) DISJOINTNESS: A classic, from Communication Complexity.

(8) ROLL CALL: Introduced in [1] (essentially), as a witnesss of $\mathsf{2D} \setminus \mathsf{1D}$.

(9) EQUAL ENDS: Introduced in [26, Prop. 2], as a problem which has small general 2DFAs but no small single-pass 2DFAs.

(10) SHORT EQUAL ENDS: Introduced in [26, Prop. 1] (essentially), as a problem which has small single-pass 2DFAs but no small 1NFAs.

(11) LENGTH: Introduced in [21, Prop. 4], as a problem against which 1NFAs are forced to stay essentially as large as 1DFAs.

(12) ONE-WAY LIVENESS: Introduced in [24, §2.1], as the first $\mathsf{1N}$-complete problem.

(13) SEPARABILITY: Introduced in [26, p. 1], as a problem that has small 1NFAs but no small single-pass 2DFAs, and is also conjectured to have no small general 2DFAs.

(14) ZERO-SAFE PROGRAMS: Introduced in [26, p. 2] (essentially), as a problem which seems "easier" than SEPARABILITY but still hard enough to have no small 2DFAs.

(15) RELATIONAL MATCH: Introduced in [19], for describing a conjecture which is equivalent to COMPACT TWL $\notin \mathsf{2D}$.

(16) RELATIONAL PATH: Introduced in [10] (essentially), as a problem which witnesses the exact value of the trade-off in converting 2NFAs to 1DFAs.

(17) FUNCTIONAL MATCH: Introduced in [19], as a restriction of RELATIONAL MATCH, useful for describing a conjecture equivalent to COMPACT TWL $\notin \mathsf{2D}$.

(18) FUNCTIONAL PATH: Introduced in [9, 10], as a problem which witnesses the exact value of the trade-off in converting 2NFAs or 2DFAs to 1NFAs, and 2DFAs to 1DFAs.

(19) FUNCTIONAL ZERO-MATCH: Introduced in [19], for facilitating reductions.

(20) RELATIONAL ZERO-MATCH: Introduced in [19], for facilitating reductions.

(21) TWO-WAY LIVENESS: Introduced in [24, §2.1], as the first $\mathsf{2N}$-complete problem.

(22) COMPACT TWL: Introduced in [19], to state a conjecture that implies $\mathsf{L/poly} \not\supseteq \mathsf{NL}$.

(23) ∃INCLUSION: Introduced in [4] (essentially), as a problem whose reverse can be used as 'core' for building witnesses for all $\mathsf{1}\Sigma_k \setminus \mathsf{2}\Pi_k$ and $\mathsf{1}\Pi_k \setminus \mathsf{2}\Sigma_k$, where $k \geq 2$.

(24) SHORT TWL: Introduced in [19], as a problem complete for $\mathsf{2N/poly}$ under $\leq_{\mathrm{h}}$.

References

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

[2] 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.

[3] J.-C. Birget, Two-way automata and length-preserving homomorphisms. *Mathematical Systems Theory* **29** (1996), 191–226.

[4] V. Geffert, An alternating hierarchy for finite automata. *Theoretical Computer Science* **445** (2012) 3, 1–24.

[5] V. Geffert, B. Guillon, G. Pighizzini, Two-way automata making choices only at the endmarkers. In: *Proceedings of LATA*. 2012, 264–276.

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

[7] V. Geffert, C. Mereghetti, G. Pighizzini, Complementing two-way finite automata. *Information and Computation* **205** (2007) 8, 1173–1187.

[8] J. Hromkovič, G. Schnitger, Nondeterminism versus determinism for two-way finite automata: generalizations of Sipser's separation. In: *Proceedings of ICALP*. 2003, 439–451.

[9] C. Kapoutsis, Removing bidirectionality from nondeterministic finite automata. In: *Proceedings of MFCS*. 2005, 544–555.

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

[11] C. Kapoutsis, Small sweeping 2NFAs are not closed under complement. In: *Proceedings of ICALP*. 2006, 144–156.

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

[13] C. Kapoutsis, Size complexity of two-way finite automata. In: *Proceedings of DLT*. 2009, 47–66.

[14] C. Kapoutsis, Two-way automata versus logarithmic space. In: *Proceedings of CSR*. 2011, 197–208.

[15] C. Kapoutsis, Nondeterminism is essential in small two-way finite automata with few reversals. *Information and Computation* **222** (2013), 208–227.

[16] C. Kapoutsis, R. Královič, T. Mömke, On the size complexity of rotating and sweeping automata. In: *Proceedings of DLT*. 2008, 455–466.

[17] C. Kapoutsis, N. Lefebvre, Analogs of Fagin's Theorem for small nondeterministic finite automata. In: *Proceedings of DLT*. 2012, 202–213.

[18] C. Kapoutsis, G. Pighizzini, Reversal hierarchies for small 2DFAs. In: *Proceedings of MFCS*. 2012, 554–565.

[19] C. Kapoutsis, G. Pighizzini, Two-way automata characterizations of L/poly versus NL. In: *Proceedings of CSR*. 2012, 222–233.

[20] D. C. Kozen, *Automata and computability*. Springer, 1997.

[21] A. R. Meyer, M. J. Fischer, Economy of description by automata, grammars, and formal systems. In: *Proceedings of FOCS*. 1971, 188–191.

[22] 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.

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

[24] W. J. Sakoda, M. Sipser, Nondeterminism and the size of two-way finite automata. In: *Proceedings of STOC*. 1978, 275–286.

[25] W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* **4** (1970), 177–192.

[26] J. I. Seiferas, *Untitled*, 1973. Manuscript.

[27] J. C. Shepherdson, The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development* **3** (1959), 198–200.

[28] M. Sipser, Halting space-bounded computations. *Theoretical Computer Science* **10** (1980), 335–338.

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

[30] M. Sipser, *Introduction to the theory of computation*. 2nd edition, Thomson Course Technology, 2006.

[31] A. Szepietowski, If deterministic and nondeterministic space complexities are equal for $\log\log n$ then they are also equal for $\log n$. In: *Proceedings of STACS*. 1989, 251–255.