Minicomplexity

Christos A. Kapoutsis*

LIAFA, Université Paris VII, France

Abstract. This is a talk on *minicomplexity*, namely on 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 recent advances, both in the theory itself and in its relation to Turing machine complexity. Finally, we illustrate a proof technique, which we call *hardness propagation by certificates*. The entire talk follows, extends, and advocates the Sakoda-Sipser framework.

1 Introduction

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

This distinction is also valid in 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 the 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 (see, e.g., [30, Chap. 1]), barely addressing any FA-complexity issues (as, e.g., in [30, Probs. 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 list of detached observations on the relative succinctness of FAs. We disagree. Before explaining, let us discuss another systematic neglect.

This is the systematic neglect of *two-way* FAs (2FAs, whose input head can move in either direction) in favor of *one-way* FAs (1FAs, whose input head can move only forward). Standard textbooks essentially identify FAs with 1FAs (see, e.g., [20, Chaps. 3–16]), only briefly addressing 2FAs, if at all, as a natural generalization (as, e.g., in [20, Chaps. 17–18]). As before, one might perhaps try to

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

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. Our goal in this talk is to help repair this ... public-relations disaster.

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

In Sect. 2, we present the fragment of minicomplexity which concerns 1FAs. We focus 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. The 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.

In Sect. 3, we continue to 2FAs. We focus on the Sakoda-Sipser conjecture and two stronger variants of it, recording their history and some recent advances. We then discuss alternation and the relationship to TM-complexity.

In Sect. 4, we present a technique for separating micomplexity classes, using closure properties (for upper bounds) together with *hardness propagation by certificates* (for lower bounds). To illustrate it, we outline a modular proof which implies an improvement on the main theorem of [9].

For $h \geq 0$, we let $[h]$ and $[\![h]\!]$ be $\{0, \ldots, h-1\}$ and its powerset. Our 2FAs are tuples $(S, \Sigma, \delta, q_{\mathrm{s}}, q_{\mathrm{a}})$ of a state set, an alphabet, a transition function, a start, and an accept state. Our *parallel automata* ($\mathrm{P}_2$1DFAs, $\cup_{\mathrm{L}}$1DFAs, $\cup_{\mathrm{R}}$1DFAs, $\cap_{\mathrm{L}}$1DFAs) are as in [16]. 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 $\mathfrak{L}$: *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 the problems $\mathfrak{L}^{\mathsf{R}} := (L^{\mathsf{R}}, \tilde{L}^{\mathsf{R}})$, $\neg\mathfrak{L} := (\tilde{L}, L)$,

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

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

$$\neg(\mathfrak{L}^{\mathsf{R}}) = (\neg\mathfrak{L})^{\mathsf{R}} \quad \neg\big(\bigwedge\mathfrak{L}\big) = \bigvee\neg\mathfrak{L} \quad \neg\big(\bigvee\mathfrak{L}\big) = \bigwedge\neg\mathfrak{L} \quad \big(\bigwedge\mathfrak{L}\big)^{\mathsf{R}} = \bigwedge\mathfrak{L}^{\mathsf{R}} \quad \big(\bigvee\mathfrak{L}\big)^{\mathsf{R}} = \bigvee\mathfrak{L}^{\mathsf{R}}$$

by the definitions. The *conjunctive concatenation* $\mathfrak{L} \wedge \mathfrak{L}'$ and *ordered star* $\mathfrak{L}{<}\mathfrak{L}'$ of two problems $\mathfrak{L}, \mathfrak{L}'$ are defined in [16] and [9]. Families of promise problems admit analogous operations. For more careful definitions, see [16,9].

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 end-markers $\vdash$ and $\dashv$, a feature unimportant for 1FAs but essential for 2FAs. Moreover, every instance is promised to be of this form, i.e., 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 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 $\emptyset$ 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. Moreover, this is not due to poor design; we easily see that M^{R} could not have done significantly better:

> *Proof.* Assume 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 $\alpha\, i$ and $\beta\, i$ the same, 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*, with a different member for every $h \geq 1$:

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

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

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

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

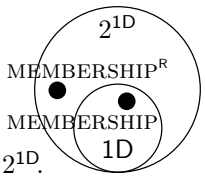
(Hence, all this time we used the terms "problem"/"algorithm" and a single description in terms of h to informally refer to and describe an entire family of promise problems/FAs. This is standard practice, which we will repeat. Also, instead of the traditional n, which is reserved for denoting input 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,

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

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 that is described in terms of h is "small" if in the implicit family of finite automata the h-th member has $\leq p(h)$ states, for some polynomial p and all h.

2.2 More Problems

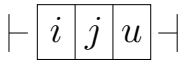
The profile of MEMBERSHIP is shared by several other elementary problems that have appeared sporadically in the literature.(1) We list some of them below. When appropriate, the endnotes explain who introduced them and why.

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:

$\vdash$ | $\check{\imath}$ | i_1 | i_2 | $\cdots$ | $\dashv$ *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.*

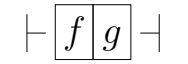
Easily, ∃EQUALITY $\in \mathsf{1D}$ but $\exists\text{EQUALITY}^{\text{R}} \in 2^{\mathsf{1D}} \setminus \mathsf{1D}$. The same holds for a variant problem, which we call SORTED ∃EQUALITY, where the numbers in the list are promised to be strictly increasing: $i_1 < i_2 < \cdots < i_l$.(2)

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 h-tuples over $[h]$:



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

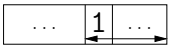
The second problem uses the alphabet ($[h] \rightharpoonup [h]$) of all partial functions on $[h]$:



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

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

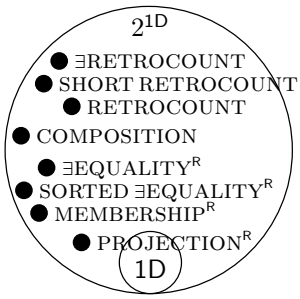
Finally, a classic. We call it RETROCOUNT, for the obvious reason:[5]



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 holds 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 their reverses are in 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 still classify problems in terms of hardness, and use the resulting lattice to deduce algorithms and lower bounds.

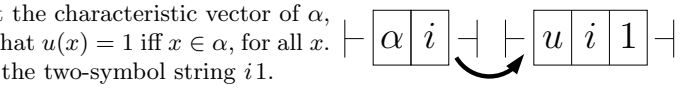


2.3 Reductions

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$^{\mathsf{R}}_h$:

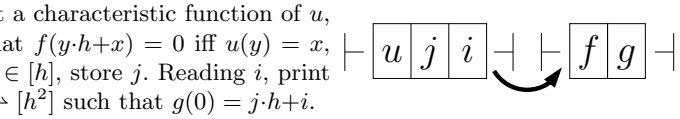
Reading $\alpha \subseteq [h]$, print the characteristic vector of α, namely $u \in [2]^h$ such that $u(x) = 1$ iff $x \in \alpha$, for all x. Reading $i \in [h]$, print the two-symbol string $i1$.



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 does not increase 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 $u\,j\,i$ maps to an instance $f g$ which satisfies $f(g(0)) = 0$ iff $u(j) = i$. Also, 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 is a family of 1DFTs $(T_h)_{h\geq 1}$ and two polynomial functions s, 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, for all x:

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

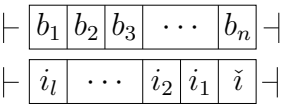
In the special case where $s(h) = 1$ for all h, every T_h is nothing more that just a mapping from symbols to strings. We then say $\mathcal{L}$ *homomorphically reduces to* $\mathcal{L}'$ and write $\mathcal{L} \leq_{\mathrm{h}} \mathcal{L}'$. Hence, by our two algorithms above, we have already shown:

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

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:

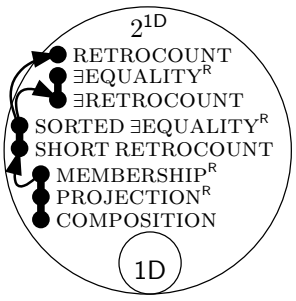
$$\mathcal{L} \leq_{\mathsf{1D}} \mathcal{L}' \quad \& \quad \mathcal{L}' \leq_{\mathsf{1D}} \mathcal{L}'' \implies \mathcal{L} \leq_{\mathsf{1D}} \mathcal{L}''\,. \tag{3}$$

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. Note 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 $s_1(h){\cdot}s_2(e_1(h))$.

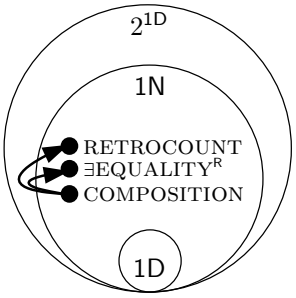
By transitivity and a few more reductions, we arrive at the lattice on the right, where arrows and lines denote $\leq_{1D}$ in one and both directions, respectively. So, by the closures and this lattice, all observations of Sect. 2.2 now follow from the three facts MEMBERSHIP$^R \notin$ 1D and ($\exists$)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



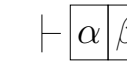
An analogy is now clear, between 1D, 2^{1D}, and $\leq_{1D}$ on one hand, and settings of TM-complexity on the other: P, EXP, and polynomial-time reductions; or L, PSPACE, and logarithmic-space reductions. So, a natural next question is whether this map also contains an analog of NP and NL.

2.4 Nondeterminism

The class 1N is defined as in (2), but for 1NFAs. To place it on our map, we note that 1D $\subseteq$ 1N $\subseteq 2^{1D}$ (by the definitions and 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.



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



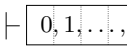
Given two sets $\alpha, \beta \subseteq [h]$, check that $\alpha \subseteq \beta$.

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.* Assume 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 $\alpha_i \alpha_j$. By $i > j$, some 1 in the representation of i is a 0 for j, so $\alpha_i \not\subseteq \alpha_j$. So, 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 INCLUSION $\leq_{1D}$ DISJOINTNESS (easily); so, 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)



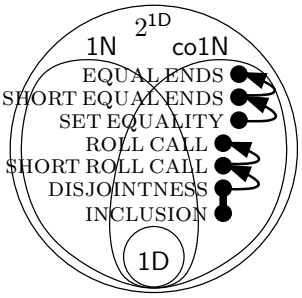
Given a list of numbers from $[h]$,
check that every number appears at least once.

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 $\mathsf{1N}$, as 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 $\neg\mathfrak{L}_h$". So, 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 additional 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, all problems from Sect. 2.2 are in $\mathsf{1N} \cap \mathsf{co1N}$: this follows, e.g., because ($\exists$) RETROCOUNT and its complement homomorphically reduce to each other (by flipping the bits) and $\mathsf{co1N}$ is closed under $\leq_{\mathsf{1D}}$ (since $\mathsf{1N}$ is).



At this point, 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 $2^{\mathsf{1D}} \setminus (\mathsf{1N} \cup \mathsf{co1N})$. We postpone the answers for Sect. 2.6 and Sect. 2.7. In the meantime, the next section examines $\mathsf{1N} \cap \mathsf{co1N}$ a bit more carefully.

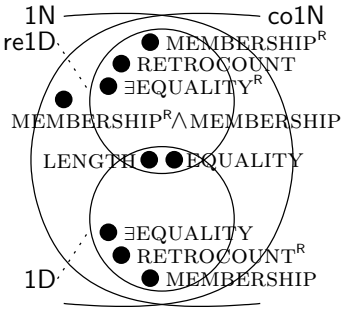
2.5 Complement and Reversal

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

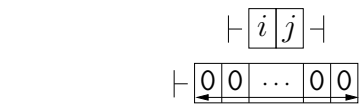
$$\mathsf{co1D} = \mathsf{1D} \neq \mathsf{re1D} \qquad \text{and} \qquad \mathsf{co1N} \neq \mathsf{1N} = \mathsf{re1N}\,,$$

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 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. Sect. 1) of any two problems that witness the two sides of the symmetric difference of $\mathsf{1D}$ and $\mathsf{re1D}$, e.g., MEMBERSHIP$^{\mathrm{R}}$ and MEMBERSHIP:

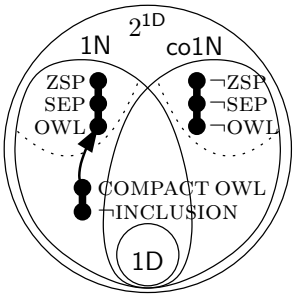


this defies small 1DFAs in either direction (as each of the original problems $\leq_{\mathrm{h}}$-reduces to it), but small 1NFAs can solve both it and its complement (easily). As for $\mathsf{1D} \cap \mathsf{re1D}$, consider any elementary problem which is the reverse of itself:



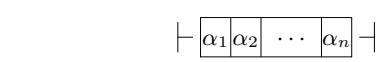
We call these two problems EQUALITY and LENGTH, respectively.[11]

2.6 Completeness

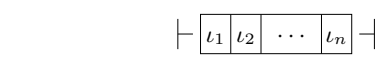


A problem is 1N-*complete* if it is in 1N and every problem in 1N reduces to it. This notion was introduced by Sakoda and Sipser [24] together with the first proof of 1N-completeness, for a problem which we call ONE-WAY LIVENESS (or just OWL).[12] A few years earlier, Seiferas [26] had introduced two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

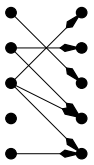
For SEPARABILITY, the alphabet is $[\![h]\!]$. To define it, we use the notion of a 'block': a string of sets $\alpha_0\alpha_1\cdots\alpha_l$ in which the first one contains the number of those that follow, i.e., $\alpha_0 \ni l$. E.g., {0,2,4}∅{1,4} and {0,2} are blocks, but ∅ and {0,2,4}{1,4} are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

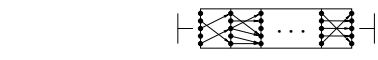


For ZERO-SAFE PROGRAMS, the alphabet is $[h] \cup [h]^2$. A symbol ι is seen as an instruction, applied to a variable $\alpha \subseteq [h]$: an $i \in [h]$ means "remove i from α"; a $(j,i) \in [h]^2$ means "if $j \in \alpha$, then add i to α". A string $\iota_1\iota_2\cdots\iota_n$ is seen as a program.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



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.





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 [14], 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, we need to 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 $\tilde{\alpha}$ of all $i \in [h]$ such that the input will be separable if it has exactly i more symbols. At start, $\tilde{\alpha} = \{0\}$, because the empty string is separable. Then, on reading an $\alpha \subseteq [h]$, we update $\tilde{\alpha}$ as follows: (1) every $0 \neq i \in \tilde{\alpha}$ is replaced by $i{-}1$, because we just read 1 symbol, and (2) if $0 \in \tilde{\alpha}$, then 0 is replaced by all $i \in \alpha$, because in the 'thread' encoded by 0 this α starts a new block. In the end, the input is accepted if $0 \in \tilde{\alpha}$.

Our 1-state 1DFT maps an instance $\alpha_1\alpha_2\cdots\alpha_n$ of SEP_h to an instance $p_1p_2\cdots p_n$ of ZSP_{2h} which uses the set variable to keep track of the overhang set $\tilde{\alpha}$. Each p_k is a sub-program $f(\alpha_k)$ which applies to $\tilde{\alpha}$ the updates (1)-(2) caused by α_k. For (1), we use instruction pairs $(i, i{-}1)i$, which replace i by $i{-}1$ if $i \in \tilde{\alpha}$, listing them by increasing i so that newly-added values do not interfere with pre-existing ones:

$$(1,0)1 \quad (2,1)2 \quad \cdots \quad (h{-}1, h{-}2)h{-}1\,. \qquad (*)$$

For (2), we can use one instruction $(0, i)$ for each $i \in \alpha_k$, and a final instruction 0 to remove 0. But there is a problem. These instructions must precede $(*)$, because all tests for $0 \in \tilde{\alpha}$ must precede $(1, 0)$, which may add 0 to $\tilde{\alpha}$; but then, placing the instructions before $(*)$, causes interference between the values added by them and the values tested for by $(*)$.

This is why we reduce to ZSP_{2h}, and not just ZSP_h: to use a set variable with both a 'lower half' in $[h]$ and an 'upper half' in $[2h]\setminus[h]$. This way, we can implement (2) 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 $\tilde{\alpha}$, '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 every $i \in \alpha_k$ to transpose the copy of α_k into the lower half, leaving the upper half empty again.

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

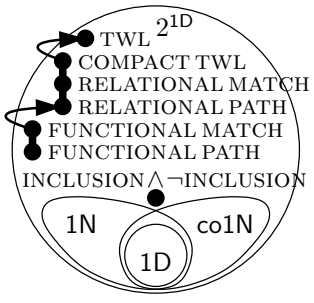
$$f(\alpha) := \quad \big((0, h{+}i)\big)_{i\in\alpha}0 \quad (1,0)1\ (2,1)2\ \cdots\ (h{-}1, h{-}2)h{-}1 \quad \big((h{+}i, i)h{+}i\big)_{i\in\alpha}$$

to produce a program $f(x)$. By our discussion above, x is separable iff its overhang set contains 0 in the end, which holds iff $f(x)$ transforms $\{0\}$ into a superset of $\{0\}$. In turn, this holds iff $f(x)$ is 0-safe (for the 'only if' direction, note that starting with a superset of $\{0\}$ cannot shrink any of the intermediate sets).

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

2.7 Harder Problems

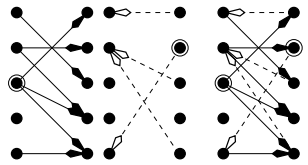
For a problem in $2^{1\mathsf{D}} \setminus (1\mathsf{N} \cup \mathsf{co}1\mathsf{N})$, we may consider the conjunctive concatenation of any two problems from the two sides of the symmetric difference of $1\mathsf{N}$ and $\mathsf{co}1\mathsf{N}$, e.g., INCLUSION $\wedge$ $\neg$INCLUSION: this is $\leq_{\text{h}}$-harder than both INCLUSION and $\neg$INCLUSION, but still in $2^{1\mathsf{D}}$. 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 it is 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 tagged. A string $iA\check{B}\check{\jmath}$ is seen again as the overlay for $A\check{B}$, but now with the i-th left-column node and the j-th right-column node marked respectively as 'entry' and 'exit'.



$\vdash \boxed{i}\boxed{A}\boxed{\check{B}}\boxed{\check{\jmath}} \dashv$ *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 a entry-to-exit path.*

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

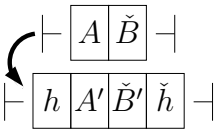
To place these problems on the map, note that FUNCTIONAL MATCH $\notin \mathsf{co1N}$ (because $\neg$DISJOINTNESS $\leq_{\mathsf{1D}}$-reduces to it, by a 1DFT which maps $\alpha\beta$ to $A\check{B}$ for $A := \{(i,i) \mid i \in \alpha\}$, $B := \{(i,i) \mid i \in \beta\}$) and that FUNCTIONAL PATH $\notin \mathsf{1N}$ [10]. Hence, both problems are outside $\mathsf{1N} \cup \mathsf{co1N}$, provided that they reduce to each other. Indeed they do, but to prove so we need two new, variant concepts.

First, the variant problem FUNCTIONAL ZERO-MATCH. This asks only that A and B '0-*match*', i.e., that their overlay contains a cycle through the 0-th left-column node.[19] This is known to be $\leq_{\mathrm{h}}$-equivalent to FUNCTIONAL MATCH [19].

Second, a variant '*nondeterministic* one-way polynomial-size reduction', $\leq_{\mathsf{1N}}$. This differs from $\leq_{\mathsf{1D}}$ 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, $\leq_{\mathsf{1N}}$ is also transitive and $\mathsf{1N}$ is closed under it.

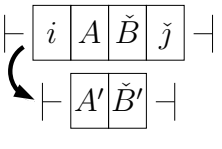
We now prove FUNCTIONAL MATCH and FUNCTIONAL PATH $\leq_{\mathsf{1N}}$-equivalent. First, we replace the first problem by its equivalent FUNCTIONAL ZERO-MATCH. Then, FUNCTIONAL ZERO-MATCH$_h$ reduces to FUNCTIONAL PATH$_{h+1}$ by a 1-state 1DFT which adds a fresh node h which is both entry and exit, 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.



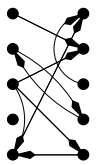
Conversely, FUNCTIONAL PATH$_h$ reduces to FUNCTIONAL ZERO-MATCH$_{h+1}$ by the simple h-state 1NFT which first transposes $A, \check{B}$ from $[h]$ to $[h{+}1] \setminus \{0\}$, then connects the 0-th left-column node to the transposed entry (via the 0-th right-column node), and the transposed exit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

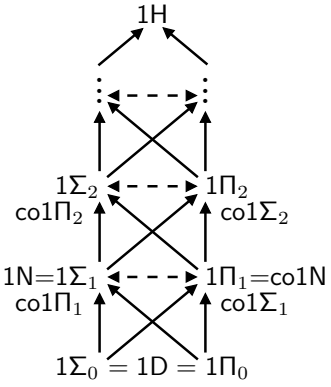
Appropriately adjusted, both of the above reductions work even when A and B are relations. Hence, RELATIONAL MATCH and RELATIONAL PATH are also $\leq_{1\mathsf{N}}$-equivalent, through the variant RELATIONAL ZERO-MATCH of the first problem, which is again known to be $\leq_{\mathrm{h}}$-equivalent to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



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

2.8 Alternation

The classes $1\mathsf{N}$ and $\mathsf{co1N}$ are on the first level of a *one-way polynomial-size hierarchy* which is defined by analogy to the polynomial-time hierarchy and the space alternating hierarchies of TM-complexity.

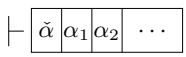


For each $k \geq 1$, we define the class $1\mathsf{\Sigma}_k$ (resp., $1\mathsf{\Pi}_k$) as in (2) but for $1\Sigma_k$FAs ($1\Pi_k$FAs), namely for alternating 1FAs which perfrom $< k$ alternations between existential and universal states, starting with an existential (universal) one. We also let $1\mathsf{\Sigma}_0, 1\mathsf{\Pi}_0 := 1\mathsf{D}$ and $1\mathsf{H} := \bigcup_{k\geq 0}(1\mathsf{\Sigma}_k \cup 1\mathsf{\Pi}_k)$. Then

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

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

The natural question is whether this hierarchy of classes is strict. For the first level, we already know (Sect. 2.4) that, e.g., ¬INCLUSION $\in 1\mathsf{\Sigma}_1 \setminus 1\mathsf{\Pi}_1$ and INCLUSION $\in 1\mathsf{\Pi}_1 \setminus 1\mathsf{\Sigma}_1$. 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. Sect. 1), we build the following table of witnesses for all levels, where 'INCL' abbreviates 'INCLUSION' (for $k = 1$, reversal is redundant; we use it only for symmetry):

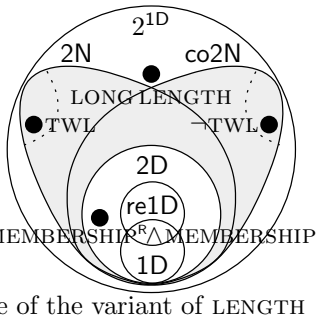
$1\mathsf{\Sigma}_1 \setminus 1\mathsf{\Pi}_1$	$1\mathsf{\Sigma}_2 \setminus 1\mathsf{\Pi}_2$	$1\mathsf{\Sigma}_3 \setminus 1\mathsf{\Pi}_3$	$1\mathsf{\Sigma}_4 \setminus 1\mathsf{\Pi}_4$	$1\mathsf{\Sigma}_5 \setminus 1\mathsf{\Pi}_5$	$\cdots$
¬INCL$^{\mathrm{R}}$	∃INCL$^{\mathrm{R}}$	$\bigvee$¬∃INCL$^{\mathrm{R}}$	$\bigvee\bigwedge$ ∃INCL$^{\mathrm{R}}$	$\bigvee\bigwedge\bigvee$¬∃INCL$^{\mathrm{R}}$	$\cdots$
INCL$^{\mathrm{R}}$	¬∃INCL$^{\mathrm{R}}$	$\bigwedge$ ∃INCL$^{\mathrm{R}}$	$\bigwedge\bigvee$¬∃INCL$^{\mathrm{R}}$	$\bigwedge\bigvee\bigwedge$ ∃INCL$^{\mathrm{R}}$	$\cdots$
$1\mathsf{\Pi}_1 \setminus 1\mathsf{\Sigma}_1$	$1\mathsf{\Pi}_2 \setminus 1\mathsf{\Sigma}_2$	$1\mathsf{\Pi}_3 \setminus 1\mathsf{\Sigma}_3$	$1\mathsf{\Pi}_4 \setminus 1\mathsf{\Sigma}_4$	$1\mathsf{\Pi}_5 \setminus 1\mathsf{\Sigma}_5$	$\cdots$

(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 Sect. 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 note 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]). So, 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, because of, e.g., MEMBERSHIP^R∧MEMBERSHIP (cf. Sect. 2.5). The last inclusion is also strict, because of the variant of LENGTH (cf. Sect. 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 Sect. 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 $\leq_h$-equivalent (Sect. 2.6). So, 2D contains either all three or none of them (because it is closed under $\leq_h$ [24,15]). The next three problems reduce to each other under $\leq_{\mathsf{1D}}$ or $\leq_{\mathsf{1N}}$ (Sect. 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 (because 2D is also closed under $\leq^{\text{lac}}_{\mathsf{2D}}$ [19]).

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

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

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

So, the Sakoda-Sipser conjecture is concretely reformulated as $\text{TWL} \notin \mathsf{2D}$, i.e.:

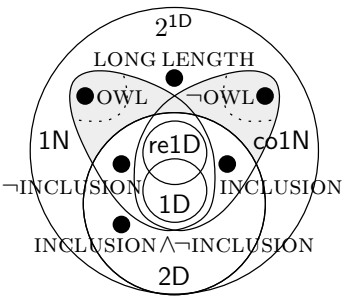
> *no* poly(h)*-state* 2DFA *can check that an* h*-tall, two-way, multi-column graph contains a path from its leftmost to its rightmost column.* (6)

Similarly, the conjecture that $\mathsf{2N} \neq \mathsf{co2N}$ has the concrete reformulation that $\neg\text{TWL} \notin \mathsf{2N}$, 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 OWL $\notin$ 2D, i.e., that (6) holds even if the graph is one-way. Since OWL is 1N-complete, this is equivalent to 1N $\not\subseteq$ 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 it for some subclass of 2DFAs of restricted bidirectionality or restricted information. A chronological list follows.

- In [26], a fairly simple argument confirmed the claim for *single-pass* 2DFAs, that is, 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, that is, 2DFAs which reverse their head only on end-markers. The structure and tools of that proof have since been copied and reused several times (see, e.g., [13,16,9]). 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, that is, 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 [13], a computability argument confirmed the claim for 2DFA *moles*, that is, 2DFAs which explore the multi-column graph of an instance of OWL as a 'system of tunnels'. In fact, the proof showed that OWL_5 is already unsolvable by 2DFA moles of any size. Hence, it also completely avoided the full claim.
- In [9], a recent argument confirmed the claim for 2DFAs *with few reversals*, that is, 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 [9], unrestricted small 2DFAs are strictly more powerful. In fact, more recent arguments show that few-reversal 2DFAs necessarily perform only $O(1)$ reversals [18, Thm. 1], and that an infinite hierarchy of computational power exists below them [18, Thm. 2]. In Sect. 4 we use these arguments to give a simpler, modular argument for improving the main theorem of [9].

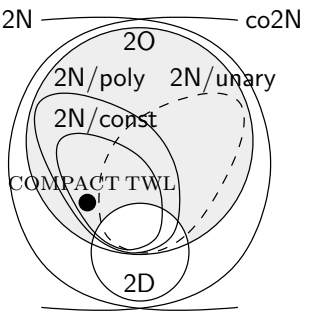
In conclusion, if the full claim OWL $\notin$ 2D is false, then this can only be by a multi-pass 2DFA algorithm which uses the full information of every symbol and, infinitely often, performs $\Theta(n)$ reversals and exhibits $\Omega(n)$ trajectories.

A similar strengthening is also possible for the second conjecture of Sect. 3.1: we believe that even ¬OWL $\notin$ 2N, i.e., that (7) holds even if the graph is one-way. One advance in this direction concerns 2NFAs of restricted bidirectionality:

- In [12], it was confirmed that ¬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, 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 have length constant, polynomial, or exponential in h, respectively. E.g., 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, involving *outer-nondeterministic* 2FAs (2OFAs, i.e., 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 2O $\subseteq$ 2DSIZE($2^{\mathrm{poly}(\log h)}$) (a theorem of [5], which uses [25]). So, COMPACT TWL admits quasi-polynomially large 2DFAs because all problems in 2O do.

We note that, like 2N/poly, the class 2N/unary of all unary problems of 2N is also in 2O (by [6]), and 2D contains either subclass iff it contains the entire 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 2O a common complete problem, for appropriate reductions under which 2D is closed [19]). Moreover, 2O = co2O (another theorem of [5]), so that this entire discussion takes place inside 2N $\cap$ co2N.

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

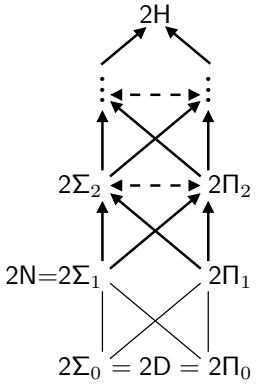
$$\text{COMPACT TWL} \notin \mathsf{2D} \iff \text{RELATIONAL MATCH} \not\leq_{\mathrm{h}} \text{FUNCTIONAL MATCH}\,. \tag{9}$$

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:

$$\begin{array}{l}\textit{no pair of systematic ways of replacing } h\textit{-tall relations}\\ \textit{by } \mathrm{poly}(h)\textit{-tall functions respects the existence of cycles.}\end{array} \tag{10}$$

So, 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 Sect. 3.5.

3.4 Alternation Again



The *two-way polynomial-size hierarchy* is defined as in Sect. 2.8 but for 2FAs. With the same witnesses as for 1FAs, we know this hierarchy does not collapse either [4]. However, 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 $2\Sigma_0 \subseteq 2\Sigma_1$ is strict,
- whether the inclusion $2\Pi_0 \subseteq 2\Pi_1$ is strict, and
- whether $2\Sigma_1$ and $2\Pi_1$ are incomparable.

Second, although the first of these questions is indeed the Sakoda-Sipser conjecture, the last two are *not* about 2D vs. co2N and 2N vs. co2N: for $k \geq 1$, it is open whether $\mathsf{co2}\Sigma_k = 2\Pi_k$ (and thus also whether $2\Sigma_k = \mathsf{co2}\Pi_k$). In fact, Geffert [4, §7] conjectures that $\mathsf{co2}\Pi_1$ is not even in 2H, and thus co2H $\neq$ 2H.

3.5 Relation to Turing Machine Complexity

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

$$
\begin{array}{ccccccc}
 & & & & \mathsf{2D} \supseteq \mathsf{2N/const} & \Longrightarrow & \mathsf{2D} \ni \text{COMPACT TWL} \\
 & & & & \Uparrow & & \Uparrow \\
\mathsf{L} \supseteq \mathsf{NL} & \Longrightarrow & \mathsf{L/poly} \supseteq \mathsf{NL} & \overset{[2,15]}{\Longleftrightarrow} & \mathsf{2D} \supseteq \mathsf{2N/poly} & \overset{[15,19]}{\Longleftrightarrow} & \mathsf{2D} \ni \text{SHORT TWL} \\
\Uparrow{\scriptstyle [31]} & & & & \Uparrow & & \Uparrow \\
\mathsf{LL} \supseteq \mathsf{NLL} & \Longrightarrow & \mathsf{LL/polylog} \supseteq \mathsf{NLL} & \overset{[15]}{\Longleftrightarrow} & \mathsf{2D} \supseteq \mathsf{2N/exp} & \overset{[15]}{\Longleftrightarrow} & \mathsf{2D} \ni \text{LONG TWL} \\
 & & & & \Uparrow & & \Uparrow \\
 & & & & \mathsf{2D} \supseteq \mathsf{2N} & \overset{[24]}{\Longleftrightarrow} & \mathsf{2D} \ni \text{TWL}
\end{array}
$$

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

The diagram can be interpreted in two ways. On one hand, people interested in space-bounded TMs can see that 2D vs. 2N offers a unifying setting for studying L vs. NL. Confirming 2D $\not\supseteq$ 2N could be a first step in a gradual attack towards 2D $\not\supseteq$ 2N/exp and 2D $\not\supseteq$ 2N/poly, hence LL $\neq$ NLL and L $\neq$ NL. Or, confirming 2D $\not\ni$ COMPACT TWL (or its combinatorial version (10)) could be a single step to L $\neq$ NL. On the other hand, people stydying 2D vs. 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. E.g., replacing 2N by its counterpart 2A for alternating 2FAs, we get L/poly $\supseteq$ P $\Longleftrightarrow$ 2D $\supseteq$ 2A/poly [15], since alternating log-space coincides with deterministic polynomial time.

4 Hardness Propagation by Certificates

We now present a modular method of separating minicomplexity classes [16,18]. This has two parts, one for upper bounds and one for lower bounds.

The first part consists in proving 'closure lemmas' of the form: *if an s-state* FA *of type X solves problem $\mathfrak{L}$, then a* poly(s)*-state* FA *of type X solves problem $\mathfrak{L}'$*, where $\mathfrak{L}'$ is derived from $\mathfrak{L}$ by via some problem operator (cf. Sect. 1). E.g., if 1UFAs are the unambiguous 1NFAs, then a straightforward closure lemma is:

Lemma 1. *If an s-state* 1UFA *solves $\mathfrak{L}$, then a $O(s)$-state* 1UFA *solves $\bigwedge\mathfrak{L}$.*

Intuitively, such a lemma says that type X can absorb the 'increase in hardness' caused by the operator which derives $\mathfrak{L}'$ from $\mathfrak{L}$.

In the second part, our approach proves 'hardness-propagation lemmas', of the form: *if no* poly(s)*-state* FA *of type X solves problem $\mathfrak{L}$, then no s-state* FA *of type X' solves problem $\mathfrak{L}'$*, where type X' is more powerful than X. E.g., a straightforward lemma involving parallel automata (cf. Sect. 1) is [9, Fact 12]:

Lemma 2. (a) *If no $\cup_{\mathrm{L}}$*1DFA *with s-state components solves $\mathfrak{L}$, then no $\cup_{\mathrm{R}}$*1DFA *with s-state components solves $\mathfrak{L}^{\mathrm{R}}$.* **(b)** *If no $\cap_{\mathrm{L}}$*1DFA *with $(s{+}1)$-state components solves $\mathfrak{L}$, then no $\cup_{\mathrm{L}}$*1DFA *with s-state components solves $\neg\mathfrak{L}$.*

Intuitively, every such lemma describes a 'propagation of hardness' from X vs. $\mathfrak{L}$ to X' vs. $\mathfrak{L}'$. Typically, we prove the contrapositive. Assuming an s-state FA M' of type X' for $\mathfrak{L}'$, we find in M' a class of objects (e.g., tuples of states, crossing sequences) which can serve as 'certificates' for the positive instances of $\mathfrak{L}$, in the sense that an instance of $\mathfrak{L}$ is positive iff it has such a certificate; then, we build a poly(s)-state FA of type X which solves $\mathfrak{L}$ by simply searching for certificates.

As an example, we prove that $\mathsf{2D}[O(1)] \not\supseteq \mathsf{1U}$, where $\mathsf{1U}$ is the restriction of $\mathsf{1N}$ to problems solvable by small 1UFAs, and $\mathsf{2D}[O(1)]$ is the restriction of $\mathsf{2D}$ to problems solvable by small 2DFAs *with $O(1)$ reversals* [18]. (By [18, Thm. 1], this strengthens the recent [9, Thm. 1].) As witness, we use the problem:

$$\mathcal{R} = (\mathfrak{R}_h)_{h\geq 1} := \bigwedge\big[\, \big(\bigwedge\textsc{membership}^{\mathrm{R}}\big) < \big(\bigwedge\textsc{membership}\big)\,\big]\,.$$

More concretely, an instance of $\mathfrak{R}_h$ is a list of the form $\$y_1\$\cdots\$y_l\$$; each y_j is a list of the form $*x_1*\cdots*x_l*$; and each x_j is a list of the form $\#\alpha_1 i_1\#\cdots\#\alpha_l i_l\#$ or $\#i_1\alpha_1\#\cdots\#i_l\alpha_l\#$. The task is to check that, in every y_j: either every x_j has some i_j not in the adjacent α_j; or x_j of both forms exist with all their i_j in the adjacent α_j, and those in set-number form precede those in number-set form.

The upper bound of this theorem, $\mathcal{R} \in \mathsf{1U}$, follows by the easy facts that $\textsc{membership}, \textsc{membership}^{\mathrm{R}} \in \mathsf{1U}\cap\mathsf{co1U}$, and by Lemma 1 and the next lemma:

Lemma 3. (a) *If s-state* 1UFAs *solve $\mathfrak{L}$,$\neg\mathfrak{L}$, then a $O(s)$-state* 1UFA *solves $\bigvee\mathfrak{L}$.* **(b)** *If s-state* 1UFAs *solve $\mathfrak{L}_{\mathrm{L}}$,$\neg\mathfrak{L}_{\mathrm{L}}$, $\mathfrak{L}_{\mathrm{R}}$,$\neg\mathfrak{L}_{\mathrm{R}}$, then a $O(s)$-state* 1UFA *solves $\mathfrak{L}_{\mathrm{L}}{<}\mathfrak{L}_{\mathrm{R}}$.*

The lower bound, $\mathcal{R} \notin \mathsf{2D}[O(1)]$, uses Lemma 2 and the additional hardness-propagation Lemmas 4–6 below, which are [16, Lemma 5], [18, Lemma 4*], and (an extension of) [18, Lemma 6*]. For each of them, we outline a proof, describing only the certificates and their usage. For the full arguments, definitions, and notation, see [16,18]. The lower bound itself is proved in the end.

Lemma 4. *If no s-state* 1DFA *solves $\mathfrak{L}$, then no* $\cap_{\text{L}}$1DFA *with s-state components solves $\bigvee\mathfrak{L}$.*

Proof. Suppose some $\cap_{\text{L}}$1DFA $M = (\mathcal{A}, \emptyset)$ solves $\bigvee\mathfrak{L}$ with k s-state components, where k is minimum possible. Pick any $D_* \in \mathcal{A}$. Let $M' = (\mathcal{A}', \emptyset) := (\mathcal{A}\backslash\{D_*\}, \emptyset)$. Since k is minimum, M' does not solve $\bigvee\mathfrak{L}$, neither does any $\cap_{\text{L}}$1DFA that differs from M' only in the selection of final states. By [16, Lemma 2], some string $y = \#x_1\#\cdots\#x_l\#$ is *confusing* for M' on $\bigvee\mathfrak{L} = (K, \tilde{K})$, namely:

$$\text{or} \quad \begin{array}{lcl} y \in K & \& & (\exists D \in \mathcal{A}')\big(D(y) = \bot\big) \\ y \in \tilde{K} & \& & (\forall D \in \mathcal{A}')(\exists \tilde{y} \in K)\big(D(y) = D(\tilde{y})\big)\,. \end{array}$$

We know $y \in \tilde{K}$. [Otherwise, $y \in K$ and some $D \in \mathcal{A}'$ hangs on it, so M does not accept y, so it does not solve $(K, \tilde{K})$, contradiction.] We also know $D_*(y) \neq \bot$. [Otherwise, $D_*(yx\#) = \bot$ for any positive x, as well, hence M does not accept $yx\# \in K$, so it does not solve $(K, \tilde{K})$, contradiction.] Let $p_* := D_*(y)$.

Definition. A state q of D_* is a *certificate* for an instance x of $\mathfrak{L}$ if it satisfies: (i) $\text{LCOMP}_{D_*,p_*}(x)$ hits right into q and (ii) D_* from q on # moves to a final state.

Claim. An instance of $\mathfrak{L}$ is positive iff it has a certificate.

Hence, to solve $\mathfrak{L}$, an s-state 1DFA simulates D_* from p_* and checks that $\dashv$ is reached in a state q from which D_* would move to a final state if it read #. □

Lemma 5. *If no* $\cup_{\text{L}}$1DFA *with* $1+\binom{s}{2}$*-state components solves $\mathfrak{L}_{\text{L}}$ and no* $\cup_{\text{R}}$1DFA *with* $1+\binom{s}{2}$*-state components solves $\mathfrak{L}_{\text{R}}$, then no* $\text{P}_2$1DFA *with s-state components solves $\mathfrak{L}_{\text{L}}{<}\mathfrak{L}_{\text{R}}$.*

Proof. Let $\mathfrak{L}_{\text{L}} = (L_{\text{L}}, \tilde{L}_{\text{L}})$, $\mathfrak{L}_{\text{R}} = (L_{\text{R}}, \tilde{L}_{\text{R}})$. Suppose some $\text{P}_2$1DFA $M = (\mathcal{A}, \mathcal{B}, F)$ solves $\mathfrak{L}_{\text{L}}{<}\mathfrak{L}_{\text{R}}$ with s-state components. Let ϑ be a *generic* string for M over the strings $L := \{\#x_1\#\cdots\#x_l\# \mid l \geq 0 \ \ \& \ \ (\forall i)(x_i \in \tilde{L}_{\text{L}} \cup \tilde{L}_{\text{R}})\}$ of negatives of $\mathfrak{L}_{\text{L}}$,$\mathfrak{L}_{\text{R}}$.

Definition. A pair $\{p, q\}$ of distinct states in M is a *forward certificate* for an instance x of $\mathfrak{L}_{\text{L}}$ or $\mathfrak{L}_{\text{R}}$ if there exists $D \in \mathcal{A}$ such that

$$p, q \in Q^D_{\text{LR}}(\vartheta) \quad \text{and} \quad \begin{array}{l} \text{if both } \text{LCOMP}_{D,p}(x\vartheta) \text{ and } \text{LCOMP}_{D,q}(x\vartheta) \text{ hit right,} \\ \text{then they do so into the same state.} \end{array}$$

A *backward certificate* is defined symmetrically, with $\mathcal{A}$, Q^D_{LR}, $\text{LCOMP}_{D,.}(x\vartheta)$, and "hit right" replaced respectively by $\mathcal{B}$, Q^D_{RL}, $\text{RCOMP}_{D,.}(\vartheta x)$, and "hit left".

Claim. At least one is true: (i) an instance of $\mathfrak{L}_{\text{L}}$ is positive iff it has a forward certificate, or (ii) an instance of $\mathfrak{L}_{\text{R}}$ is positive iff it has a backward certificate.

If (i) is true, then an instance x of $\mathfrak{L}_{\text{L}}$ is positive iff there is $D \in \mathcal{A}$ and distinct $p, q \in Q^D_{\text{LR}}(\vartheta)$ such that *either* one of $\hat{c}_p := \text{LCOMP}_{D,p}(x\vartheta)$ or $\hat{c}_q := \text{LCOMP}_{D,q}(x\vartheta)$ hangs *or* both hit right into the same state. Hence, to solve $\mathfrak{L}_{\text{L}}$, a $\cup_{\text{L}}$1DFA checks this condition using one $1+\binom{s}{2}$-state component $D_{p,q}$ for every such combination

of D and p, q. On input x, $D_{p,q}$ runs a synchronized simulation of the prefixes $c_p := \textsc{lcomp}_{D,p}(x)$ and $c_q := \textsc{lcomp}_{D,q}(x)$ of $\hat{c}_p$ and $\hat{c}_q$. If at any point c_p,c_q are about to enter the same state or one of them is about to hang, then $D_{p,q}$ enters a special state $\top$ which consumes the rest of x and accepts. Otherwise, c_p,c_q hit right into distinct states p',q'; then $D_{p,q}$ accepts iff one of $\textsc{lcomp}_{D,p'}(\vartheta)$ or $\textsc{lcomp}_{D,q'}(\vartheta)$ hangs or they both hit right into the same state.

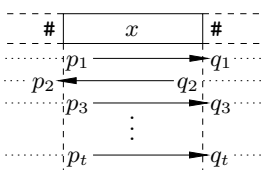
If (ii) is true, we work symmetrically with $\mathfrak{L}_{\mathrm{R}}$ and backward certificates. □

Lemma 6. *If no* $\textsc{p}_2$1DFA *with s-state components solves $\mathfrak{L}$, then no s-state* 2DFA *with $O(1)$ reversals solves $\bigwedge\mathfrak{L}$.*

Proof. Let $\mathfrak{L} = (L, \tilde{L})$. Let M be an s-state 2DFA with $O(1)$ reversals for $\bigwedge\mathfrak{L}$. Then M performs $< r_*$ reversals on every input of length $> n_*$, for some r_*, n_*. Let $Q = \{0, \ldots, s{-}1\}$ be the state set of M, and let $m_* := \max(r_*, n_*)$.

Pick any $x \in L$. Then $w := \#(x\#)^{m_*}$ is a positive of $\bigwedge\mathfrak{L}$. So, $c := \textsc{comp}_M(w)$ is accepting. Since $|w| \geq m_*{+}1 > n_*$, the reversals in c are $< r_* \leq m_*$, hence fewer than the copies of x in w. So, on some of these copies, c performs 0 reversals.

Fix any of the copies with 0 reversals. On it, c consists of $t \leq 2s$ one-way traversals (one-way, since there are 0 reversals; and $\leq 2s$, or else c would repeat a state on the first cell of x, and loop). Let $\overline{p}_x := (p_1, \ldots, p_t)$ and $\overline{q}_x := (q_1, \ldots, q_t)$ be the crossing sequences of c on the outer boundaries of the particular copy of x.



Let $\mathcal{C} := \{(\overline{p}_x, \overline{q}_x) \mid x \in L\}$ be all crossing-sequence pairs created like this.

Definition. A pair $(\overline{p}, \overline{q})$ of t-long sequences of states of M is a *certificate* for an instance x of $\mathfrak{L}$ if (i) it is in $\mathcal{C}$, and

(ii) For every odd $i = 1, \ldots, t$: $\textsc{lcomp}_{M,p_i}(x)$ is one-way and hits right into q_i.

(iii) For every even $i = 1, \ldots, t$: $\textsc{rcomp}_{M,q_i}(x)$ is one-way and hits left into p_i.

Claim. An instance of $\mathfrak{L}$ is positive iff it has a certificate.

Hence, to solve $\mathfrak{L}$, a $\textsc{p}_2$1DFA $P := (\{A_p \mid p \in Q\}, \{B_p \mid p \in Q\}, F)$ searches for a certificate. Each component A_p (resp., B_p) simulates M from p for as long as it moves right (left); if M ever attempts to reverse, the component hangs. Thus, on input x, P simulates M from every state and in either fixed direction, covering all possible one-way traversals of x. In the end, P checks whether x has a certificate by comparing the results of these $2s$ computations against every $(\overline{p}, \overline{q}) \in \mathcal{C}$. Formally, for each $\overline{p} = (p_1, \ldots, p_t)$ and $\overline{q} = (q_1, \ldots, q_t)$ we let $F_{(\overline{p},\overline{q})}$ be the set of all $2s$-tuples that we can build from two copies of Q

$$(\ 0, 1, \ldots, s{-}1,\ \ 0, 1, \ldots, s{-}1\ ),$$

by replacing (i) every odd-indexed p_i in the left copy with the respective q_i (so that A_{p_i} hits right into q_i); (ii) every even-indexed q_i in the right copy with the respective p_i (so that B_{q_i} hits left into p_i) and (iii) all other states in either copy with any result in $Q \cup \{\bot\}$ (to let all other 1DFAs free). Thus, $F_{(\overline{p},\overline{q})}$ is all tuples which prove that $(\overline{p}, \overline{q})$ is a certificate. Finally, we let $F := \bigcup_{(\overline{p},\overline{q})\in\mathcal{C}} F_{(\overline{p},\overline{q})}$. □

We now prove that $\mathcal{R} \notin \mathsf{2D}[O(1)]$. This follows by applying Lemmas 2, 4–6 to the fact that $\neg\text{MEMBERSHIP}^{\text{R}} \notin \mathsf{1D}$. For brevity, we let $\mathfrak{M}_h := \text{MEMBERSHIP}_h$.

1. No (2^h-2)-state 1DFA solves $\neg\mathfrak{M}_h^{\text{R}}$, by a proof similar to that of Sect. 2.1.
2. No $\cap_{\text{L}}$1DFA with (2^h-2)-state components solves $\bigvee\neg\mathfrak{M}_h^{\text{R}}$, by 1 and Lemma 4.
3. No $\cup_{\text{L}}$1DFA with (2^h-3)-state components solves $\bigwedge\mathfrak{M}_h^{\text{R}}$, by 2 and Lemma 2b.
4. No $\cup_{\text{R}}$1DFA with (2^h-3)-state components solves $\bigwedge\mathfrak{M}_h$, by 3 and Lemma 2a.
5. Every $\text{P}_2$1DFA for $\bigwedge\mathfrak{M}_h^{\text{R}} < \bigwedge\mathfrak{M}_h$ has at least one $\Omega(2^{h/2})$-state component, by 3, 4, and Lemma 5.
6. Every $O(1)$-reversal 2DFA for $\mathfrak{R}_h$ has $\Omega(2^{h/2})$ states, by 5 and Lemma 6.

This proves the lower bound for $\mathcal{R}$, completing the proof that $\mathsf{2D}[O(1)] \not\supseteq \mathsf{1U}$.

5 Conclusion

This was an introduction to the *complexity of two-way finite automata*, or *minicomplexity*. We presented it within the Sakoda-Sipser framework, to emphasize the tight analogy with standard TM-complexity. We believe that this view helps reveal important structure, which otherwise passes unnoticed. This is, of course, a coarse view, which is unable to distinguish beyond polynomial differences. For finer views, at the level of asymptotic or exact values, one should resort to the more standard vocabulary in terms of 'trade-offs'.

We focused on one-way and two-way heads and on deterministic, nondeterministic, and alternating modes. However, minicomplexity also includes other heads (rotating, sweeping) and other standard modes from TM-complexity (probabilistic, quantum, interactive); see [14] for a broader view. It can also mimic TM-complexity in other ways. E.g., see [17] for a first step towards *descriptive minicomplexity*, where minicomplexity classes receive logical characterizations.

The selection of the presented material reflects this author's immediate interests and 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] concerning alternating 2FAs.

References

1. B. H. Barnes. A two-way automaton with fewer states than any equivalent one-way automaton. *IEEE Transactions on Computers*, C-20(4):474–475, 1971.
2. P. Berman and A. Lingas. On complexity of regular languages in terms of finite automata. Report 304, Institute of Computer Science, Polish Academy of Sciences, Warsaw, 1977.
3. J.-C. Birget. Two-way automata and length-preserving homomorphisms. *Mathematical Systems Theory*, 29:191–226, 1996.
4. V. Geffert. An alternating hierarchy for finite automata. *Theoretical Computer Science*. To appear.

5. V. Geffert, B. Guillon, and G. Pighizzini. Two-way automata making choices only at the endmarkers. In *Proceedings of LATA*, pages 264–276, 2012.
6. V. Geffert, C. Mereghetti, and G. Pighizzini. Converting two-way nondeterministic unary automata into simpler automata. *Theoretical Computer Science*, 295:189–203, 2003.
7. V. Geffert, C. Mereghetti, and G. Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007.
8. J. Hromkovič and G. Schnitger. Nondeterminism versus determinism for two-way finite automata: generalizations of Sipser's separation. In *Proceedings of ICALP*, pages 439–451, 2003.
9. C. Kapoutsis. Nondeterminism is essential in small two-way finite automata with few reversals. *Information and Computation*. To appear.
10. C. Kapoutsis. Removing bidirectionality from nondeterministic finite automata. In *Proceedings of MFCS*, pages 544–555, 2005.
11. C. Kapoutsis. *Algorithms and lower bounds in finite automata size complexity*. Phd thesis, Massachusetts Institute of Technology, June 2006.
12. C. Kapoutsis. Small sweeping 2NFAs are not closed under complement. In *Proceedings of ICALP*, pages 144–156, 2006.
13. C. Kapoutsis. Deterministic moles cannot solve liveness. *Journal of Automata, Languages and Combinatorics*, 12(1-2):215–235, 2007.
14. C. Kapoutsis. Size complexity of two-way finite automata. In *Proceedings of DLT*, pages 47–66, 2009.
15. C. Kapoutsis. Two-way automata versus logarithmic space. In *Proceedings of CSR*, pages 197–208, 2011.
16. C. Kapoutsis, R. Královič, and T. Mömke. On the size complexity of rotating and sweeping automata. In *Proceedings of DLT*, pages 455–466, 2008.
17. C. Kapoutsis and N. Lefebvre. Analogs of Fagin's Theorem for small nondeterministic finite automata. In *Proceedings of DLT*, 2012. To appear.
18. C. Kapoutsis and G. Pighizzini. Reversal hierarchies for small 2DFAs. Submitted.
19. C. Kapoutsis and G. Pighizzini. Two-way automata characterizations of L/poly versus NL. In *Proceedings of CSR*, pages 222–233, 2012.
20. D. C. Kozen. *Automata and computability*. Springer, 1997.
21. A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *Proceedings of FOCS*, pages 188–191, 1971.
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(10):1211–1214, 1971.
23. M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959.
24. W. J. Sakoda and M. Sipser. Nondeterminism and the size of two-way finite automata. In *Proceedings of STOC*, pages 275–286, 1978.
25. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.
26. J. I. Seiferas. Untitled. Manuscript, Oct. 1973.
27. J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3:198–200, 1959.
28. M. Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10:335–338, 1980.
29. M. Sipser. Lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences*, 21(2):195–202, 1980.

30. M. Sipser. *Introduction to the theory of computation.* Thomson Course Technology, 2nd edition, 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*, pages 251–255, 1989.

Notes

(1) MEMBERSHIP: Introduced in [16, p. 459], as a problem whose reverse can be used as 'core' for building witnesses of separations of complexity classes. See also [9, 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 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 appears to be "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 [11] (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 [10,11], 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], for stating 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 $\leq_{\mathrm{h}}$.