Minicomplexity

Some Motivation, Some History, and Some Structure (Invited Talk Extended Abstract)

Christos A. Kapoutsis(✉)

Carnegie Mellon University in Qatar, Doha, Qatar
cak@cmu.edu

Abstract. The term *minicomplexity* was first suggested in [2], as a name for the field of theory of computation which studies the *size complexity of two-way finite automata*, as outlined in [1]. In this talk, we discuss the motivation behind this field and enumerate some of its prominent results in their historical context. By reformulating these results, we then attempt to reveal additional structure which often passes unnoticed. The present report records the start of this attempt.

1 Machines vs. Machines

Central in [1] is the invitation to start viewing the results in this field similarly to how results are being viewed in standard complexity theory: not as statements about the relative power of various computational devices, but as statements about the relative difficulty of various computational problems.

To describe the difference between the two viewpoints and stress the benefits of such a shift, we go back to the seminal paper of Meyer and Fischer [5], which initiated the field, and to the three very first propositions in it. The first one(1)

> **Proposition 1.** For every $n > 0$, there is a regular set $R_n \subset \{0,1\}^*$ such that the reduced finite automaton accepting R_n has exactly 2^n states (and size 2^{n+1}), but there is an n-state nondeterministic finite automaton of size $3n-2$ accepting R_n.

says that one-way nondeterministic finite automata (1NFAs) are strictly more powerful than deterministic ones (1DFAs), as some binary witness language R_n needs $\leq n$ states on 1NFAs but $\geq 2^n$ states on 1DFAs. In the proof, R_n is described only through its deciding 1NFA. After it, one more witness R'_n is given:

> An even simpler example suggested by Paterson is close to optimal: let $R'_n \subset \{0,1\}^*$ be the set of strings whose n^{th} from the last digit is 1. Then R'_n is recognized by an $n+1$ state nondeterministic machine, the reduced deterministic machine has 2^n states, and the reduced machine for the reversal of R'_n has $n+2$ states. Moreover, $R'_n \cap \{0,1\}^n\{0,1,\lambda,\}^n$ is a finite event with similar properties.

B. Catania et al. (Eds.): SOFSEM 2019, LNCS 11376, pp. 28–38, 2019.
https://doi.org/10.1007/978-3-030-10801-4_3

which is (suboptimal, but) simpler, together with its restriction R''_n (our name) to strings of length $<2n$, which shares the same properties.(2) The next proposition

> **Proposition 2.** For every $n > 1$, there is a regular set $F_n \subset \{0,1,2\}^*$ which can be recognized by a two-way finite automaton with at most $5n+5$ states, but the reduced finite automaton accepting F_n has at least n^n states. Moreover F_n is a finite set.
>
> **Proof.** Let $F_n = \{0^{i_1} 1 0^{i_2} 1 \cdots 1 0^{i_n} 2^k 0^{i_k} \mid 1 \le k \le n \text{ and } 1 \le i_j \le n \text{ for } j = 1,\ldots,n\}$.

establishes a similar relation for two-way deterministic finite automata (2DFAs) and 1DFAs: now a witness language F_n needs $O(n)$ states on 2DFAs but $\geq n^n$ states on 1DFAs. Finally, the third proposition

> **Proposition 3.** For each $n > 0$ there is a regular event $P_n \subset \{0,1,2\}^*$ which is recognizable by a deterministic one pebble automaton with at most $3n+5$ states. The reduced finite automaton accepting P_n has at least 2^{2^n} states. Moreover, P_n is finite.
>
> **Proof.** $P_n = \{x_1 2 x_2 2 \ldots 2 x_k 2 2 x_i \mid x_j \in \{0,1\}^n$ for $1 \le j \le k$, $i \le k$, and regarded as binary integer $x_j < x_{j+1}$ for $1 \le j < k\}$. □

proves that, augmented with a pebble, 2DFAs are even more powerful: $O(n)$ states are now enough to decide a language P_n for which 1DFAs need $\geq 2^{2^n}$ states. In short, and in the standard parlance of the field, Propositions 1–3 tell us that the trade-off in the conversion from 1NFAs, 2DFAs, or single-pebble 2DFAs to 1DFAs is respectively $2^{\Omega(n)}$, $2^{\Omega(n \lg n)}$, and $2^{2^{\Omega(n)}}$.

Overall, this is entirely a "*machines vs. machines*" discussion: computational devices compete against each other, and we want to know which is more powerful. In this competition, problems play only the secondary role of a witness task on which a stronger machine beats a weaker one by solving it with less resources.

2 Problems vs. Problems

The alternate viewpoint is "*problems vs. problems*": computational tasks compete against each other, and we want to know which is more difficult. This time it is machines that play the secondary role, of a witness device on which a harder problem beats an easier one by requiring more resources.

This is the viewpoint of standard complexity theory, and it was proposed for minicomplexity, as well, by Sakoda and Sipser in their own seminal paper [6]. By switching to this viewpoint, we bring problems at the center of attention: we clearly describe them as computational tasks (as opposed to sets of strings); give them distinctive names; and collect them in complexity classes relative to the

various machines and the polynomiality or not (as opposed to the asymptotics) of the used resources. For practical reasons, we also use h as the important parameter (instead of n, which is often needed as input length); and describe the instances over a large alphabet and with an associated promise for the format (if this simplifies the description without affecting the difficulty).

2.1 RETROCOUNT

For example, R'_h (Proposition 1) is the problem: "Given a bitstring, check that its h-th from the last digit is 1." Note that the reference "h-th from the last digit" is void on strings of length $<h$. We could get into a discussion of whether such strings should be accepted or not, but this would be a distraction from the main point of the task. A better description of the essence of this computational problem is one where the intended format of the input is taken for granted:

$$\text{"Given a bitstring of length } \geq h, \text{ check that its } h\text{-th last bit is } 1." \tag{1}$$

so that a solving machine need not check that the length is $\geq h$; if it is not, then the machine is free to decide arbitrarily. We call this problem RETROCOUNT_h. Similarly, the restriction R''_h is the problem $\text{SHORT RETROCOUNT}_h$:

$$\text{"Given a bitstring of length } h \leq n < 2h, \text{ check that its } h\text{-th last bit is } 1." \tag{2}$$

Again, it is not the job of a solving machine to check that the length of the input is appropriate; this is promised. The job is only to check the h-th last bit.

Formally, a (*promise*) *problem* $\mathfrak{L}$ over an alphabet Σ is a pair $(L, \tilde{L})$ of disjoint subsets of Σ^*. A string w is an *instance* of $\mathfrak{L}$ if $w \in L \cup \tilde{L}$, 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 $w \in \tilde{L}$ (behaving arbitrarily on $w \notin L \cup \tilde{L}$). So, (1) and (2) are the promise problems

$$\begin{aligned} \text{RETROCOUNT}_h &:= \big(\quad \{0,1\}^* 1 \{0,1\}^{h-1}, \quad \{0,1\}^* 0 \{0,1\}^{h-1} \quad \big) \\ \text{SHORT RETROCOUNT}_h &:= \big(\ \{0,1\}^{<h} 1 \{0,1\}^{h-1}, \ \{0,1\}^{<h} 0 \{0,1\}^{h-1} \ \big). \end{aligned}$$

That both problems witness the exponential difference in number of states between 1NFAs and 1DFAs is expressed by the fact that both belong to the class

$$\mathsf{1N} := \left\{ (\mathfrak{L}_h)_{h\geq 1} \;\middle|\; \begin{array}{l} \text{there exist 1NFAs } (N_h)_{h\geq 1} \text{ and polynomial } s \\ \text{such that every } N_h \text{ solves } \mathfrak{L}_h \text{ with } s(h) \text{ states} \end{array} \right\}, \tag{3}$$

of "problems solved by small 1NFAs", but not in the respective class 1D for 1DFAs:

$$\text{RETROCOUNT}, \ \text{SHORT RETROCOUNT} \in \mathsf{1N} \setminus \mathsf{1D}, \tag{4}$$

where $\text{RETROCOUNT} = (\text{RETROCOUNT}_h)_{h\geq 1}$ is the induced family, and similarly for SHORT RETROCOUNT. Note that, since the latter problem is a restriction of the former, all the information of (4) follows from the next two facts and lemma.

Fact 1. $\text{RETROCOUNT} \in \mathsf{1N}$.

Fact 2. SHORT RETROCOUNT $\notin$ 1D.

Lemma 1. *If* $\mathcal{L} \subseteq \mathcal{L}'$ *and* $\mathcal{L} \notin \mathsf{C}$, *then* $\mathcal{L}' \notin \mathsf{C}$.

Here, $\mathcal{L} \subseteq \mathcal{L}'$ means that the family $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1}$ is a *restriction* of $\mathcal{L}' = (\mathfrak{L}'_h)_{h\geq 1}$ (so that SHORT RETROCOUNT $\subseteq$ RETROCOUNT); equivalently, we also say that $\mathcal{L}'$ is a *generalization* of $\mathcal{L}$. Formally, for $\mathfrak{L} = (L, \tilde{L})$ and $\mathfrak{L}' = (L', \tilde{L}')$, we write $\mathfrak{L} \subseteq \mathfrak{L}'$ if both $L \subseteq L'$ and $\tilde{L} \subseteq \tilde{L}'$; then we write $\mathcal{L} \subseteq \mathcal{L}'$ if $\mathfrak{L}_h \subseteq \mathfrak{L}'_h$ for all h.

2.2 PROJECTION

For another example, F_h (Proposition 2) is the problem: "Given a string, check that it consists of a tuple of h numbers from 1 to h (in unary, by 0s; delimited by 1s), an index k from 1 to h (in unary, by 2s), and the k-th number in the tuple (in unary, by 0s)." Clearly, the essence of this task is to check that the number after k equals the k-th number in the tuple. So, a better description is:

> "Given a tuple of h numbers from 1 to h (in unary, by 0s; delimited by 1s), an index k from 1 to h (in unary, by 2s), and a number i from 1 to h (in unary, by 0s), check that i equals the k-th number in the tuple." (5)

so that, as above, checking that the input is correctly formatted is not important.

Also unimportant is the fact that the numbers and index are in unary. The problem preserves its essence, if we assume that the input cells are large enough to host any number from $[h] := \{1, \dots, h\}$. So, an even better description of F_h is

> "Given a tuple $i_1, i_2, \dots, i_h$ of numbers in $[h]$, and two numbers k, i in $[h]$, check that i equals i_k." (6)

where the input alphabet is $[h]$. Now it is more clear what the problem is: to check that the projection of the given tuple to its k-th component returns i. So, we refer to (6) as PROJECTION$_h$;[(3)] and use UNARY PROJECTION$_h$ for (5).

With these clarifications, Proposition 2 says that UNARY PROJECTION $\in$ 2D $\setminus$ 1D, where the class 2D of "problems solved by small 2DFAs" is defined similarly to (3). Intuitively, the reasons for this fact are clear: When a solving 1DFA crosses the boundary between the tuple and the index, it must be able to answer any query of the form "does the k-th component equal i?", so it must store the full tuple, which needs $\geq h^h$ states. In contrast, a 2DFA can read $2^k 0^i$ and store k and i; rewind; then countdown to the k-th block of 0s to check that it contains exactly i of them, all doable with $O(h^2)$ states. Similarly, PROJECTION $\in$ 2D $\setminus$ 1D.

Now that we see why these problems witness 2D $\setminus$ 1D, we may further ask: Do we really need the tuple numbers in separate cells? Or k and i in separate cells? No. Over the alphabet $[h]^h \cup [h]^2$, where cells are large enough to host an entire h-tuple $\bar{i}$ or query $u = (k, i)$, we may define the problem COMPACT PROJECTION$_h$:

> "Given a tuple $\bar{i} \in [h]^h$ and a query $u \in [h]^2$, check that $u_2 = i_{u_1}$." (7)

Intuitively, this is the best description of the essence of F_h, as it contains exactly the structure that is sufficient and necessary to place it in 2D $\setminus$ 1D.

Of course, problems (5), (6), and (7) are "essentially the same". To describe this intuition formally, we first define them as promise problems. E.g., (7) is:

$$\textsc{compact projection}_h := \big(\{\bar{i}u \mid\ u_2 = i_{u_1}\},\ \{\bar{i}u \mid\ u_2 \neq i_{u_1}\}\big);$$

and similarly for (5) and (6), over alphabets $\{0,1,2\}$ and $[h]$. We then introduce reductions, as follows. For arbitrary problems $\mathfrak{L} = (L, \tilde{L})$ and $\mathfrak{L}' = (L', \tilde{L}')$ over alphabets Σ and Σ', we say that $\mathfrak{L}$ $\mathsf{1D}$-*reduces* to $\mathfrak{L}'$ ($\mathfrak{L} \leq_{\mathsf{1D}} \mathfrak{L}'$) if there exists a one-way deterministic finite transducer (1DFT) T such that

$$w \in L \implies T(w) \in L' \qquad \text{and} \qquad w \in \tilde{L} \implies T(w) \in \tilde{L}' \tag{8}$$

where $T(w)$ is the output of T on input w, if T accepts w, or undefined, otherwise. An alternative and more concise way to write (8) is:

$$T(\mathfrak{L}) \subseteq \mathfrak{L}' \tag{9}$$

where $T(\mathfrak{L}) = T(L, \tilde{L}) = (T(L), T(\tilde{L})) = \big(\{T(w) \mid w \in L\}, \{T(w) \mid w \in \tilde{L}\}\big)$ is the pair of the images under T of all positive and all negative instances of $\mathfrak{L}$ (which is itself a problem iff $T(L) \cap T(\tilde{L}) \neq \emptyset$). As further alternative,

$$\mathsf{1D}(\mathfrak{L}) \subseteq \mathfrak{L}' \tag{10}$$

says the same, without identifying T (i.e., it is equivalent to $\mathfrak{L} \leq_{\mathsf{1D}} \mathfrak{L}'$).

In the special case where the inclusion (9) is an equality, $\mathfrak{L}'$ is a $\mathsf{1D}$-*image* of $\mathfrak{L}$ under T, and we also write (10) as equality. E.g., (5) is a $\mathsf{1D}$-image of (6):

$$\mathsf{1D}(\textsc{projection}_h) = \textsc{unary projection}_h \tag{11}$$

via the $O(h)$-state 1DFT T which scans an instance $i_1 i_2 \dots i_h k i$ and, for each symbol in $[h]$, prints the appropriate unary representation and delimiters. Note that T prints on its output tape only $h + 2 = \text{poly}(h)$ times; and, in each of these times, the printed string has length $\leq h + 1 = \text{poly}(h)$.

In another special case, where T has only 1 state and always accepts, T defines a homomorphism $H : \Sigma \cup \{\vdash, \dashv\} \to (\Sigma')^*$ such that $T(w) = H(\vdash w \dashv)$. We then say that $\mathfrak{L}$ *homomorphically reduces* to $\mathfrak{L}'$ ($\mathfrak{L} \leq_{\mathsf{H}} \mathfrak{L}'$ or $\mathsf{H}(\mathfrak{L}) \subseteq \mathfrak{L}'$), if $H(\mathfrak{L}) \subseteq \mathfrak{L}'$; or that $\mathfrak{L}'$ is a *homomorphic image* of $\mathfrak{L}$ ($\mathsf{H}(\mathfrak{L}') = \mathfrak{L}$), if $H(\mathfrak{L}) = \mathfrak{L}'$. Hence,

$$\mathsf{H}(\textsc{compact projection}_h) = \textsc{projection}_h \tag{12}$$

via the homomorphism H which maps every h-tuple $\bar{i} = (i_1, i_2, \dots, i_h)$ to the string $i_1 i_2 \cdots i_h$, every query $u = (k, i)$ to the string ki, and each of $\vdash, \dashv$ to ϵ. Note that H maps every symbol to a string of length $\leq h = \text{poly}(h)$.

These definitions extend to problem families $\mathcal{L} = (\mathfrak{L}_h)_{h \geq 1}$ and $\mathcal{L}' = (\mathfrak{L}'_h)_{h \geq 1}$, if every $\mathfrak{L}_h$ reduces to some $\mathfrak{L}'_{h'}$ via a 1DFT T_h or a homomorphism H_h. However, to say that $\mathcal{L} \leq_{\mathsf{1D}} \mathcal{L}'$, $\mathsf{1D}(\mathfrak{L}) = \mathcal{L}'$, $\mathcal{L} \leq_{\mathsf{H}} \mathcal{L}'$, or $\mathsf{H}(\mathfrak{L}) = \mathcal{L}'$, we also need h' and the size of T_h to be small relative to h: namely, that $h' = \text{poly}(h)$ and T_h has $\text{poly}(h)$ states. If, in addition, every T_h prints only $\text{poly}(h)$ times, then we write

$\mathcal{L} \leq^{\text{lac}}_{\mathsf{1D}} \mathcal{L}'$ and call the T_h *laconic*; if every printed string has length only poly(h), then we write $\mathcal{L} \leq^{\text{t}}_{\mathsf{1D}} \mathcal{L}'$ (or $\mathcal{L} \leq^{\text{t}}_{\mathsf{H}} \mathcal{L}'$) and call the T_h (or H_h) *tight*. Hence,

$$\begin{aligned} \mathsf{1D}(\text{PROJECTION}) &= \text{UNARY PROJECTION} \\ \mathsf{H}(\text{COMPACT PROJECTION}) &= \text{PROJECTION} \end{aligned} \tag{13}$$

by the tight laconic transducers of (11) and the tight homomorphisms of (12).

In conclusion, (13) expresses the intuition that problems (5), (6), and (7) are "essentially the same". Now, the fact that they all witness $\mathsf{2D} \setminus \mathsf{1D}$ follows from only two easy facts and standard lemmas [6, Sect. 3], [4, Corollary 3]:

Fact 3. UNARY PROJECTION $\in \mathsf{2D}$.

Fact 4. COMPACT PROJECTION $\notin \mathsf{1D}$.

Lemma 2. $\mathsf{2D}$ *is closed under* $\leq_{\mathsf{H}}$ *and* $\leq^{\text{lac}}_{\mathsf{1D}}$.

Lemma 3. $\mathsf{1D}$ *is closed under* $\leq_{\mathsf{1D}}$ *(and thus also under* $\leq_{\mathsf{H}}$ *and* $\leq^{\text{lac}}_{\mathsf{1D}}$*).*

2.3 MEMBERSHIP

Let us now return to Proposition 1 and see how large alphabets can help us better understand the essense of its problems, too.

In SHORT RETROCOUNT$_h$, every instance w is of the form uv, where $|u| = h$ and $0 \leq |v| < h$. Note that the actual bits of v are unimportant; only $l := |v|$ matters: w is positive iff the $(l+1)$-st bit of u is $\mathtt{1}$. Namely, if $\alpha \subseteq [h]$ is the set of the indices of all $\mathtt{1}$'s in u, then w is asking whether $l+1 \in \alpha$. So, the question is really whether a set α contains an element i; it's just that α is given in binary (by its characteristic vector u) and i is given in unary (by the length $i-1$ of v).

Let us also recall why SHORT RETROCOUNT $\in \mathsf{1N} \setminus \mathsf{1D}$. On crossing the u-v boundary, a solving 1DFA must be able to handle any l, i.e., any query of the form "does the i-th bit of u equal $\mathtt{1}$?"; so it must store the full u, which needs $\geq 2^h$ states. In contrast, a 1NFA can scan u; guess the crucial $\mathtt{1}$; countdown from h on the next bits (entering v at count i, for i the index of the guessed $\mathtt{1}$); and accept iff the count is 1 on $\dashv$ (so $|v| = i-1$), all doable with $O(h)$ states.

Now that we better understand what the problem is asking and why it is a witness, we may ask: Do we really need α in binary and i in unary? No. Over the alphabet $\{\alpha \mid \alpha \subseteq [h]\} \cup [h]$, we define the problem: "Given a set $\alpha \subseteq [h]$ and an element $i \in [h]$, check that $i \in \alpha$", or formally:[(4)]

$$\text{MEMBERSHIP}_h := \big(\{\alpha i \mid\ i \in \alpha\},\ \{\alpha i \mid\ i \notin \alpha\}\big); \tag{14}$$

and claim that this best captures the essence of SHORT RETROCOUNT$_h$. That the two problems are "essentially the same" is formally expressed by the fact that the former homomorphically reduces to the latter via the obvious homomorphism which maps every α to its characteristic vector and every i to $\mathtt{0}^{i-1}$:

$$\mathsf{H}(\text{MEMBERSHIP}) \subseteq \text{SHORT RETROCOUNT}. \tag{15}$$

What about RETROCOUNT$_h$? Its instances are derived by left-padding those of SHORT RETROCOUNT$_h$ by arbitrary bitstrings. Formally, let LPAD be the operator which maps $\mathfrak{L} = (L, \tilde{L})$ to the pair $\mathsf{LPAD}(\mathfrak{L}) := (\{0,1\}^* L, \{0,1\}^* \tilde{L})$. This pair is not necessarily a promise problem: if there exist instances $w \in L$ and $\tilde{w} \in \tilde{L}$ and pad-strings $x, \tilde{x} \in \{0,1\}^*$ such that $xw = \tilde{x}\tilde{w}$, then the two sets in the pair are not disjoint. So, call $\mathfrak{L}$ *left-paddable*, if this does not happen. Then a family $\mathcal{L} = (\mathfrak{L}_h)_{h \geq 1}$ is *left-paddable* if every $\mathfrak{L}_h$ is; and $\mathsf{LPAD}(\mathcal{L}) := (\mathsf{LPAD}(\mathfrak{L}_h))_{h \geq 1}$.

Easily, SHORT RETROCOUNT$_h$ is left-paddable, since the sign of each instance is determined by its last h bits and these are unaffected by the padding; and

$$\mathsf{LPAD}(\text{SHORT RETROCOUNT}) = \text{RETROCOUNT}. \tag{16}$$

Overall, (15) and (16) formally relate (1), (2), and (14) to each other. Now, the fact that all three witness $\mathsf{1N} \setminus \mathsf{1D}$ follows from only two easy facts and from suitable lemmas (Lemma 1, as $\mathcal{L} \subseteq \mathsf{LPAD}(\mathcal{L})$; Lemma 3; and Lemmas 4–5):

Fact 5. SHORT RETROCOUNT $\in \mathsf{1N}$.

Fact 6. MEMBERSHIP $\notin \mathsf{1D}$.

Lemma 4. 1N *is closed under* $\leq_{\mathsf{H}}$.

Lemma 5. *If* $\mathcal{L}$ *is left-paddable and* $\mathcal{L} \in \mathsf{1N}$*, then* $\mathsf{LPAD}(\mathcal{L}) \in \mathsf{1N}$.

Note how our earlier Facts 1 and 2 now become corollaries of Facts 5 and 6.

Retrocount vs. Projection. There is great similarity between our intuition why the projection problems are not in 1D and why the same holds for the retrocount problems. This suggests that the projection problems also have MEMBERSHIP at their core. Indeed:

$$\mathsf{H}(\text{MEMBERSHIP}) \subseteq \text{COMPACT PROJECTION} \tag{17}$$

by the homomorphism which maps every set $\alpha \subseteq [h]$ to its "characteristic tuple" $\bar{i} \in [h+1]^{h+1}$ where $i_j = 1$ or $h+1$, based on whether $j \in \alpha$ or not, respectively; and each $i \in [h]$ to the query $(i, 1)$.[(5)] So, our earlier Fact 4 is now a corollary of Fact 6 (via (17) and Lemma 3). At the same time, MEMBERSHIP is also a witness of $\mathsf{2D} \setminus \mathsf{1D}$, because Fact 3 implies it is in 2D (via (13), (17), and Lemma 2).

2.4 LIST MEMBERSHIP

We now continue to problem P_h (Proposition 3): "Given a string, check that it is a strictly increasing list of h-long binary numbers (delimited by 2s), followed by a copy of one of them (separated by 22)." Clearly, the essence of this task is to check that the number after 22 appears in the preceding list. The condition that the list is strictly increasing is there to ensure that P_h is finite. Ignoring it (also dropping the finiteness of P_h from Proposition 3), we arrive at this better description:

> "Given a list of h-long binary numbers (delimited by 2s) and an h-long binary number i (separated by 22), check that i is in the list." (18)

As previously, presenting the numbers in binary is unimportant; all that matters is that each block of h bits can host 2^h different strings. It is important, however, to know when we have arrived at i. So, to zoom into the essence of P_h, we switch to alphabet $[2^h] \cup \{\check{\imath} \mid i \in [2^h]\}$, where each cell hosts a full number x (possibly ticked, as $\check{x}$) in $[2^h]$ (as opposed to $\{0, \ldots, 2^h - 1\}$), and define the problem:

$$\begin{array}{l}\text{“Given a list of numbers from } [2^h] \text{ and a ticked number } i \in [2^h],\\ \text{check that } i \text{ is in the list.”}\end{array} \tag{19}$$

In it, one easily sees a variant of MEMBERSHIP$_h$, where elements are drawn (not from $[h]$, but) from $[2^h]$; and the set is given (not in a single cell, but) as a list over many cells, possibly with repetitions. To represent this problem, we first introduce its variant over the smaller alphabet $[h]$:[(6)]

$$\begin{array}{l}\text{LIST MEMBERSHIP}_h :=\\ \quad \big(\{i_1 i_2 \cdots i_t \check{\imath} \mid\ t \geq 0 \ \&\ i_1, i_2, \ldots, i_t, i \in [h] \ \&\ (\exists j)(i_j = i)\},\\ \quad \{i_1 i_2 \cdots i_t \check{\imath} \mid\ t \geq 0 \ \&\ i_1, i_2, \ldots, i_t, i \in [h] \ \&\ (\forall j)(i_j \neq i)\}\big),\end{array} \tag{20}$$

and refer to (19) itself, over $[2^h]$, as TALL LIST MEMBERSHIP$_h$. Then, for (18) we use the name BINARY TALL LIST MEMBERSHIP$_h$.

So, Proposition 3 says that BINARY TALL LIST MEMBERSHIP $\in \mathsf{P_1D} \setminus 2^{\mathsf{1D}}$, where $\mathsf{P_1D}$ and 2^{1D} are the classes for small single-pebble 2DFAs and large 1DFAs, where “large” means “with $2^{\text{poly}(h)}$ states”. Once again, the intuitive reasons are clear: on crossing 22, a solving 1DFA must have stored the set of numbers occurring in the list, which needs $\geq 2^{2^h}$ states. In contrast, a single-pebble 2DFA can compare i against every i_j bit-by-bit, using the pebble to mark the current i_j, all doable with $O(h)$ states. By the same reasons, TALL LIST MEMBERSHIP $\in \mathsf{P_1D} \setminus 2^{\mathsf{1D}}$. (Note that it is important for the pebble 2DFA to have the list spread across cells.)

Note that our intuition for the lower bound is the same as for MEMBERSHIP, except now there are exponentially more sets to remember. To represent this formally, let us first note that

$$\mathsf{H}(\text{MEMBERSHIP}) \subseteq \text{LIST MEMBERSHIP} \tag{21}$$

via the homomorphism which maps every set $\alpha \subseteq [h]$ to a string $i_1 i_2 \cdots i_t$ of its members; and every $i \in [h]$ to its ticked variant $\check{\imath}$. We then also note that (19) can be obtained from LIST MEMBERSHIP by applying an operator TALL,

$$\mathsf{TALL}(\text{LIST MEMBERSHIP}) = \text{TALL LIST MEMBERSHIP}, \tag{22}$$

which maps a family $\mathcal{L} = (\mathfrak{L}_h)_{h \geq 1}$ to its sub-family $\mathsf{TALL}(\mathcal{L}) = (\mathfrak{L}_{2^h})_{h \geq 1}$ at indices which are powers of 2; and (18) can be obtained from (19) homomorphically

$$\mathsf{H}(\text{TALL LIST MEMBERSHIP}) = \text{BINARY TALL LIST MEMBERSHIP} \tag{23}$$

by mapping every $i \in [2^h]$ to the h-long binary representation of $i - 1$, preceded or followed by 2, depending on whether i is ticked or not. Now, the lower bound of Proposition 3 follows from (21), (22), and (23) and a strengthening of Fact 6.

To see how, we start with some definitions and facts. The class quasi-1D corresponds to 1DFAs with quasi-polynomially many states (i.e., $2^{\mathrm{poly}(\log n)}$ states). We can easily show the next strengthening of Fact 6 (by the standard reasoning, that MEMBERSHIP$_h$ needs $\geq 2^h$ states on a 1DFA) and variation of Lemma 3:

Fact 7. MEMBERSHIP $\notin$ quasi-1D.

Lemma 6. quasi-1D *and* 2^{1D} *are closed under* $\leq_H$.

A family $\mathcal{L} = (\mathfrak{L}_h)_{h\geq 1}$ is *self-homomorphic* if $\mathfrak{L}_h \leq_H \mathfrak{L}_{h'}$ for all $h \leq h'$; intuitively, if the instances of every $\mathfrak{L}_h$ can be seen as instances of every higher $\mathfrak{L}_{h'}$. Easily, LIST MEMBERSHIP is self-homomorphic, and we can prove that:

Lemma 7. *If* $\mathcal{L}$ *is self-homomorphic and* $\mathcal{L} \notin$ quasi-1D, *then* $\mathsf{TALL}(\mathcal{L}) \notin 2^{1D}$.

Now, we can apply the following reasoning:

MEMBERSHIP $\notin$ quasi-1D	(Fact 7)
$\implies$ LIST MEMBERSHIP $\notin$ quasi-1D	(Lemma 6 and (21))
$\implies$ TALL LIST MEMBERSHIP $\notin 2^{1D}$	(Lemma 7 and (22))
$\implies$ BINARY TALL LIST MEMBERSHIP $\notin 2^{1D}$	(Lemma 6 and (23))

Overall, we see that the lower bound of Proposition 3 follows from the hardness of the core problem of Proposition 1 (Fact 7) and from properties of the classes.

For the upper bound of Proposition 3, we see that one of the two witnesses satisfies it because the other one does: TALL LIST MEMBERSHIP $\in \mathsf{P_1D}$ follows from the next fact and lemma, and since the homomorphisms of (23) are tight:

Fact 8. BINARY TALL LIST MEMBERSHIP $\in \mathsf{P_1D}$.

Lemma 8. $\mathsf{P_1D}$ *is closed under* $\leq^t_H$.

3 Modular Witnesses

The three propositions of [5] offered witness languages for the differences $\mathsf{1N} \setminus \mathsf{1D}$, $\mathsf{2D} \setminus \mathsf{1D}$, and $\mathsf{P_1D} \setminus 2^{1D}$. By analyzing these languages, we arrived at eight promise problems witnessing these differences. In the end, all bounds that we needed for these problems followed from Facts 3, 5, 8 (for the upper bounds) and Fact 7 (for the lower bounds), via the established relations between these problems and using a collection of lemmas of three distinct types:

– *preservation of hardness*: These are lemmas of the form

$$\mathcal{L} \notin \mathsf{C} \implies \mathcal{L}' \notin \mathsf{C}$$

where $\mathcal{L}'$ is derived from $\mathcal{L}$. E.g., Lemma 1 is such a lemma, with $\mathcal{L}'$ any generalization of $\mathcal{L}$. Similarly for every lemma for the closure of a class under $\leq_H$ or $\leq_{1D}$, with $\mathcal{L}'$ any generalization of $\mathsf{H}(\mathcal{L})$ or $\mathsf{1D}(\mathcal{L})$.

– *propagation of hardness*: These are lemmas of the form

$$\mathcal{L} \notin \mathsf{C} \implies \mathcal{L}' \notin \mathsf{C}'$$

where $\mathcal{L}', \mathsf{C}'$ are derived from $\mathcal{L}, \mathsf{C}$ [3]. E.g., Lemma 7 is such a lemma, where $\mathcal{L}' = \mathsf{TALL}(\mathcal{L})$, and $\mathsf{C}' = 2^{\mathsf{1D}}$ is derived from $\mathsf{C} = \mathsf{quasi\text{-}1D}$ by an application of the general operator which raises the size bound $f(h)$ of a class to $f(2^h)$.
– *preservation of easiness*: These are lemmas of the form

$$\mathcal{L} \in \mathsf{C} \implies \mathcal{L}' \in \mathsf{C}$$

where $\mathcal{L}'$ is derived from $\mathcal{L}$. E.g., Lemma 5 is such, with $\mathcal{L}' = \mathsf{LPAD}(\mathcal{L})$ and $\mathsf{C} = \mathsf{1N}$. Same for any lemma for the closure of a class under an operation.

Hence, the propositions of [5] are connected via structural relations between their witnesses, which are easy to miss if we do not adopt the right point of view.

We now further observe that the relations between witnesses allow us to express each of them as a generalization of a problem that can be obtained from MEMBERSHIP by applying a sequence of operators. Specifically, in the following list, every witness on the left generalizes the problem on the right:

SHORT RETROCOUNT:	$\mathcal{H}_{15}(\text{MEMBERSHIP})$
RETROCOUNT:	$\mathsf{LPAD}(\mathcal{H}_{15}(\text{MEMBERSHIP}))$
COMPACT PROJECTION:	$\mathcal{H}_{17}(\text{MEMBERSHIP})$
PROJECTION:	$\mathcal{H}_{13}(\mathcal{H}_{17}(\text{MEMBERSHIP}))$
UNARY PROJECTION:	$\mathcal{T}_{13}(\mathcal{H}_{13}(\mathcal{H}_{17}(\text{MEMBERSHIP})))$
TALL LIST MEMBERSHIP:	$\mathsf{TALL}(\mathcal{H}_{21}(\text{MEMBERSHIP}))$
BINARY TALL LIST MEMBERSHIP:	$\mathcal{H}_{23}(\mathsf{TALL}(\mathcal{H}_{21}(\text{MEMBERSHIP})))$

where $\mathcal{H}_{15}$ is the family of the homomorphic reductions that justifies (15), and similarly for all other $\mathcal{H}_i$ and $\mathcal{T}_i$. Every lower bound for a witness on the left was established by proving a lower bound for the corresponding problem on the right and then using Lemma 1. Again by (the contrapositive of) Lemma 1, every upper bound for a witness on the left is also an upper bound for the respective problem on the right. Overall, the problems on the right witness the same differences as the problems on the left.

Let a *modular witness* for a difference $\mathsf{C}' \setminus \mathsf{C}$ be any problem which belongs to the difference and is derived from MEMBERSHIP by applying a sequence of operators. Our discussion above shows that every one of the differences in Propositions 1–3 of [5] admits a modular witness.

We conjecture that the same is true for all differences in minicomplexity. Namely that, for every two minicomplexity classes C and C':

If $\mathsf{C}' \setminus \mathsf{C}$ is not empty, then it contains a modular witness.

In the talk, we will examine as evidence supporting this conjecture several examples of known separations where the offered witnesses were indeed (generalizations of) modular ones or can be replaced by modular witnesses.

If this conjecture is true, then designing a witness for a separation reduces to (i) deciding which sequence of operators to apply to MEMBERSHIP, and then (ii) proving the corresponding necessary lemmas of hardness propagation and of hardness or easiness preservation. Gradually, this could lead to a library of operators and corresponding lemmas, available for reuse in (i) and (ii). It would, of course, also be interesting to see a proof of the conjecture, that explains why MEMBERSHIP is sufficient as the only "seed of hardness" in this domain.

If the conjecture is false, it would be interesting to see examples where it fails. Understanding these examples, one could then suggest conditions under which the conjecture remains valid, and work with this updated, restricted variant.

Notes

(1) Proposition 1 also includes a last sentence, that the reverse of R_n needs only $O(n)$ states on a 1DFA. We omitted that sentence, as it is redundant for our purposes.

(2) We write "$<2n$", although the definition of R''_n also allows strings of length exactly $2n$. Excluding such strings does not change the desired properties of R''_n in the context of Proposition 1; and is convenient for our purposes.

(3) In [2], the name PROJECTION was used for the reverse of this problem.

(4) In [2], the name MEMBERSHIP was used for the reverse of this problem.

(5) Note the redundant component i_{h+1}, which is always set of $h+1$. We need $h+1$ components, because we need $h+1$ values; and we need $h+1$ values, because we need to ensure that the max and min values are distinct even when $h=1$.

(6) In [2], the reverse of this problem was called ∃EQUALITY.

References

1. Kapoutsis, C.A.: Size complexity of two-way finite automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 47–66. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02737-6_4
2. Kapoutsis, C.A.: Minicomplexity. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 20–42. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31623-4_2
3. Kapoutsis, C., Královič, R., Mömke, T.: Size complexity of rotating and sweeping automata. J. Comput. Syst. Sci. **78**(2), 537–558 (2012)
4. Kapoutsis, C., Pighizzini, G.: Two-way automata characterizations of L/poly versus NL. Theory Comput. Syst. **56**, 662–685 (2015)
5. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proceedings of FOCS, pp. 188–191 (1971)
6. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proceedings of STOC, pp. 275–286 (1978)