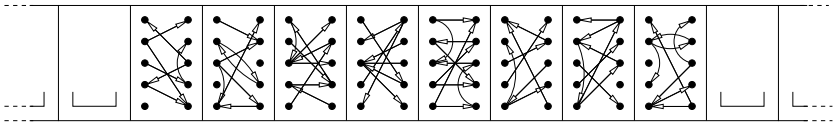
Size Complexity of Two-Way Finite Automata

Christos A. Kapoutsis

Department of Computer Science, University of Cyprus

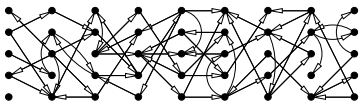
Abstract. This is a talk on the size complexity of two-way finite automata. We present the central open problem in the area, explain a motivation behind it, recall its early history, and introduce some of the concepts used in its study. We then sketch a possible future, describe a natural systematic way of pursuing it, and record some of the progress that has been achieved. We add little to what is already known—only exposition, terminology, and questions.

A problem. On the tape of a Turing machine (TM) lies an input of the form:



(1)

That is, each non-blank symbol is a two-column graph with the same, constant number of nodes per column. The question that the machine has to answer is the following: In the multi-column graph produced by identifying adjacent columns



does there exist a path from (a node of) *the leftmost column to* (a node of) *the rightmost one*? For example, in the above graph such paths exist (can you discover one?) and the answer should be "yes".

Perhaps it looks a bit strange that an entire graph fits in one tape cell. But this is not an issue. It is just that the input alphabet of this TM is a bit large: if each column has h nodes, then this alphabet consists of $2^{(2h)^2}$ symbols/graphs.

How hard is this problem? How much time/space will the TM need in order to solve it? Very little. Our problem is, in fact, regular. An easy way to prove this is to solve it with a *two-way nondeterministic finite automaton* (2NFA)—recall that 2NFAs solve exactly the regular problems [22,23,26]. Here is how:

> We start on the leftmost symbol. We nondeterministically guess a node in the left column of that symbol, and "focus" on it. From then on, we only remember which node of the current symbol we focus on, and repeat: nondeterministically guess one of the arrows out of the focused node, and make its destination the new focus. If we ever focus on a node in the rightmost column, we accept. (2)

To implement this simple algorithm, a 2NFA will need at most $2h$ states.

An alternative, more direct, but slightly more complicated way to prove the regularity of our problem is to solve it with a standard (*one-way*) *deterministic finite automaton* (1DFA). Here is how:

> We start on the leftmost symbol. We always move right. At each step, we remember for the nodes of the left column of the current symbol the following: (i) for each of them, whether it is reachable from the leftmost column via a path that lies entirely to our left, and (ii) for each pair (u, v) of them, whether v is reachable from u via a path that lies entirely to our left. On reaching a blank symbol, we accept iff the answer in (i) is "yes" for at least one node.

To implement this algorithm, a 1DFA will need at most 2^{h+h^2} states.

So, no matter which of the two algorithms it chooses to implement, our TM will solve the problem in linear time and zero space. Notice, however, the huge blow-up in the number of states that it is going to need if it decides not to use nondeterminism: instead of $2h$ states, it will need more than 2^{h^2}. And this is not due to lack of ingenuity in designing the deterministic algorithm: it can be proved that no other 1DFA can do with significantly fewer states. That is, the blow-up is unavoidable. So, if, e.g., each column has 16 nodes, then drawing the states of the 2NFA can be done on 1 page and in 1 minute, whereas drawing the states of the 1DFA would need more matter than we can see in the universe and would finish long after the sun has burnt out.[1] Nondeterminism wins.

But this comparison is unfair, one complains. The nondeterministic algorithm was allowed to use a two-way head, but the deterministic algorithm was not. For a fair comparison, the deterministic automaton should also be two-way; i.e., it should be a *two-way deterministic finite automaton* (2DFA). The more powerful head will probably help it solve the problem with much fewer states.

Good point. So, let's see. How would a 2DFA solve the problem? One's first attempt would probably be some kind of depth-first search inside the multi-column graph. But this doesn't work: it needs a stack of visited nodes which can grow arbitrarily large, and thus cannot fit in any finite number of states—let alone a small one. One would not give up so easily, though: sure, out-of-the-box depth-first search doesn't work, but certainly some other, cleverer version of graph exploration does. No it doesn't. To use significantly fewer states than the 1DFA, the 2DFA must do more than simply explore the graph [14]; it must use its bidirectionality both within the input [28] and at the two ends of the input [25]; and it must trace at least a linear (with respect to the input length) number of different trajectories [12].

In fact, nobody knows whether the minimum number of states in a 2DFA that solves our problem is closer to the $2h$ of the 2NFA or closer to the 2^{h^2} of the 1DFA. The best known lower bound is $\Omega(h^2)$ [4] and the best known upper bound is $2^{O(h^2)}$ [26]. At this "exponential" level of ignorance, the correct question is:

Can a 2DFA solve problem (1) with $p(h)$ states, for some polynomial p? (Q)

and is wide open. But, right now, it is probably some other question that mostly bothers you—a meta-question:

Who cares? ($\mathcal{Q}$)

Determinism v Nondeterminism. A central theme in the theory of computation is the comparison between deterministic and nondeterministic computations. Most characteristic in this theme is, of course, the P v NP question, a special case of the following, more general question about the *time* used by deterministic and nondeterministic *Turing machines* (DTMs and NTMs):

$$\text{Can DTMs always stay at most polynomially slower than NTMs?} \qquad (\mathrm{Q_t})$$

Less prominent, but also very important, is the L v NL question, a special case of the following, more general question about the *space* used by *Turing machines*:

$$\text{Can DTMs always use at most linearly more space than NTMs?} \qquad (\mathrm{Q_s})$$

Despite the richness and sophistication of our theory around these questions, it is probably fair to say that our progress against their core has been slow. This has led some to suspect that *the same elusive idea may lie at the center of all problems of this kind*, little affected by the particulars of the underlying computational model and resource. If this view is correct, then a possibly advantageous approach is to study restricted models of computation.

For an extreme example, consider TMs whose heads neither turn nor write. Is nondeterminism essential there? Before pondering the question, we should specify the resource under consideration. Under these restrictions, TMs are just *one-way finite automata*. So, neither time nor space is interesting, as both 1DFAs and 1NFAs use linear time and zero space.[2] Instead, observation confirms that in this case it is the *size* of the machines, as expressed by the number of states, that reveals the nondeterministic advantage. So, the analogue to $(\mathrm{Q_t})$ and $(\mathrm{Q_s})$ is:

$$\text{Can 1DFAs always stay at most polynomially larger than 1NFAs?} \qquad (\mathrm{Q_1})$$

The answer is well-known to be "no" [20]. E.g., the promise problem

$$\big(\{\alpha i \mid \alpha \subseteq [h] \text{ and } i \in \alpha\},\ \{\alpha i \mid \alpha \subseteq [h] \text{ and } i \in [h] - \alpha\} \big) \qquad (3)$$

of checking whether a set α of numbers from 1 to h contains a number i (α and i given in this order), needs only h states on 1NFAs but at least 2^h states on 1DFAs.

Hence, in this first example, the restrictions were so strong that the resulting question was easy to answer. Backing up a bit, we may now consider TMs whose heads cannot write (but can turn). Such machines are essentially identical to *two-way finite automata*. As before, observation confirms *size* as the resource that reveals the nondeterministic advantage, and the question

$$\text{Can 2DFAs always stay at most polynomially larger than 2NFAs?} \qquad (\mathrm{Q_2})$$

is our new analogue to $(\mathrm{Q_t})$ and $(\mathrm{Q_s})$.

One might expect that $(\mathrm{Q_2})$ is as easy as $(\mathrm{Q_1})$. After all, it is again about finite automata. How hard can a question about finite automata be? Automata have been studied extensively since the 1950's and the answers to most interesting questions about them are already in the textbooks, right? Not really.

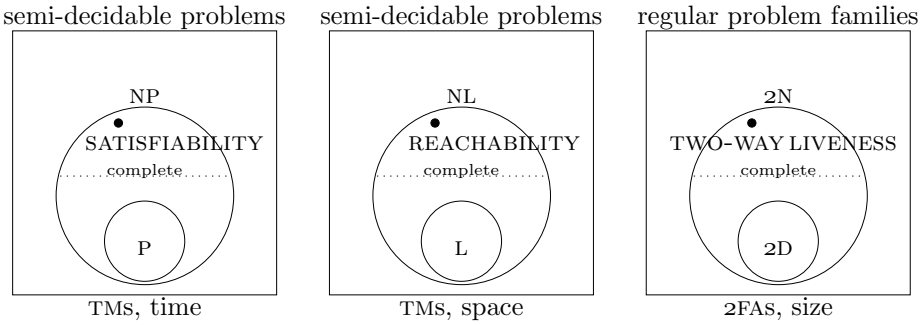


Fig. 1. An analogy between P v NP and L v NL and 2D v 2N.

Such a claim may be fair only if it refers to *computability* questions about finite automata. In contrast, *complexity* questions about finite automata have been addressed only sporadically and by relatively few researchers. Many interesting and hard questions about them remain wide open. Question (Q_2) is one of them.

Research on (Q_2) is supported by an elegant theory that mirrors the theory of NP-completeness that was developed around (Q_t) and the theory of NL-completeness that was developed around (Q_s) (Fig. 1). Proposed by Sakoda and Sipser [24] in 1978, the theory starts with the class 2D of all families of regular problems that can be solved by 2DFAs of polynomially growing size

$$\text{2D} := \left\{ (L_h)_{h\geq 1} \;\middle|\; \begin{array}{l} \text{there exist 2DFAs } (M_h)_{h\geq 1} \text{ and polynomial } p \text{ such that} \\ M_h \text{ solves } L_h \text{ with at most } p(h) \text{ states, for all } h \end{array} \right\}, \quad (4)$$

and the class 2N, defined for 2NFAs in a similar manner. For example, if C_h is problem (1) when each column has h nodes, then our discussion in the previous section proves that the family $C := (C_h)_{h\geq 1}$ is in 2N, and (Q) is asking whether it is also in 2D. Moreover, the question

$$\text{2D} = \text{2N} \; ? \qquad (Q')$$

is easily seen to be equivalent to (Q_2) in the special case of families of 2NFAs whose sizes grow polynomially.

Sakoda and Sipser went on to introduce appropriate reductions between problem families, the so-called *homomorphic reductions*, proved that 2D is closed under them, and identified a particular family in 2N that is complete with respect to them. That family was exactly C, the family whose 5th member is (1), and this is exactly how they named it—a pretty boring name, we'll call it TWO-WAY LIVENESS instead.[3] Thus, (Q) is equivalent to (Q′); it is a concrete version of the 2D v 2N problem, in the same sense that the questions

Can a DTM solve SATISFIABILITY in $p(n)$ time, for some polynomial p?
Can a DTM solve REACHABILITY in $\lg\big(p(n)\big)$ space, for some polynomial p?

(where n is the input length) are concrete versions of P v NP and L v NL (Fig. 1).

So, to return to our meta-question ($\mathcal{Q}$): One reason why one may want to care about (Q) is that it can be seen as a "microscopic version" of our big questions on the power of nondeterminism, P v NP and L v NL, a question that is simultaneously complex enough to seem relevant and simple enough to seem tractable. Conceivably, by answering (Q) we might get to understand aspects of nondeterminism which are currently inaccessible through the big questions.

In addition, the connection to L v NL is more than simply conceptual. In 1977 Berman and Lingas [1] proved that, if L = NL then (in our terminology), for a polynomial p and all h, some $p(h)$-state 2DFA decides TWO-WAY LIVENESS$_h$ correctly on every $p(h)$-long input. Hence, if we can answer (Q) in the negative using only polynomially long instances, then we can also prove L $\neq$ NL—an exciting connection, which should nevertheless be received with reserve: establishing a negative answer via exponentially long strings appears to be hard already.

Much like P v NP and L v NL, most people believe that 2D $\neq$ 2N, as well.

A stronger conjecture. The possibility 2D $\neq$ 2N had actually been conjectured earlier than [24,1] and more strongly. In a 1973 manuscript [25], J. Seiferas had conjectured that sometimes a 2NFA can stay super-polynomially smaller than all 2DFAs *even without turning its head*. That is, he had conjectured that even 1NFAs can solve problems with super-polynomially fewer states than 2DFAs.

Seiferas went on to suggest a few such problems. In one of them, the input alphabet is all sets of numbers from 0 to $h-1$. E.g., if $h=8$, then the string

$$\{1,2,4\}\emptyset\{4\}\{0,4\}\{2,4,6\}\{4\}\{4,6\}\emptyset\{3,6\}\emptyset\{2,4\}\{5,7\}\{0,3\}\{4,7\}\emptyset\{4\}\emptyset\{4\}\{0,1\}\{2,5,6\}\{1\} \tag{5}$$

is an input. A substring $\alpha_0\alpha_1\cdots\alpha_l$ of sets forms a *block* if the first set contains the number of sets after it, i.e., if $\alpha_0 \ni l$. The question is: *Can the input be separated into blocks*? E.g., the answer for (5) is "yes" because of the separation

$$\{1,\mathbf{2},4\}\emptyset\{4\}\ \ \{0,\mathbf{4}\}\{2,4,6\}\{4\}\{4,6\}\emptyset\ \ \{\mathbf{3},6\}\emptyset\{2,4\}\{5,7\}\ \ \{\mathbf{0},3\}\ \ \{4,\mathbf{7}\}\emptyset\{4\}\emptyset\{4\}\{0,1\}\{2,5,6\}\{1\}$$

where indeed the first set in each substring contains the number of sets after it in the substring, as indicated by boldface. In contrast, the answer for the string $\{1,2,7\}\{4\}\{5,6\}\emptyset\{3,6\}\{2,4,6\}$ is "no", as there is (easily) no way to break it into blocks. Seiferas called the set of all separable strings L_h—another boring name, we'll call it SEPARABILITY$_h$ instead, and let SEPARABILITY := (SEPARABILITY$_h$)$_{h\geq 1}$.

Solving this problem nondeterministically is straightforward and cheap. A 1NFA can implement the following algorithm with only h states:

> We scan the input from left to right. At the start of each block, we read the first set. If it is empty, we just hang (in this nondeterministic branch). Otherwise, we nondeterministically select from it the correct number of remaining sets in the block. We then consume as many sets, counting from that number down. When the count reaches 0, we know the block is over and a new one will follow. In the end, we accept if the input and our last count-down finish together. (6)

Seiferas conjectured that, in contrast, no 2DFA can solve SEPARABILITY$_h$ with $p(h)$ states, for any polynomial p.[4]

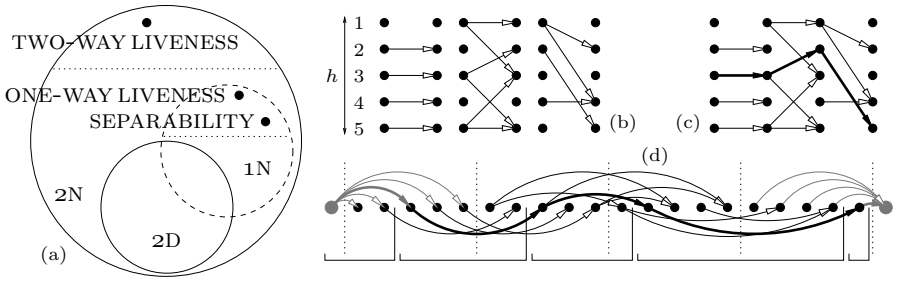


Fig. 2. (a) 1N in the map of 2D v 2N. (b) Symbols from the alphabet of ONE-WAY LIVENESS $_5$; (c) the multi-level graph they define; in bold: a live path; (d) the same graph, "flattened" for the purposes of the reduction to SEPARABILITY$_9$; dashed vertical lines distinguish the four columns; in grey: the extra nodes and arrows; in bold: a path connecting the extra nodes; also shown: the blocks defined by this path.

Sakoda and Sipser agreed with this stronger conjecture. In their terminology, this could be written as 2D $\not\supseteq$ 1N, where the class 1N is defined as in (4) but for 1NFAs (Fig. 2a). They also identified a problem family that is 1N-complete with respect to homomorphic reductions: the restriction of TWO-WAY LIVENESS to symbols/graphs with only left-to-right arrows. They called that restriction B—names were really boring in the 70's, we'll call it ONE-WAY LIVENESS (Fig. 2bc). Of course, completeness implied that 2D $\not\supseteq$ 1N $\iff$ ONE-WAY LIVENESS $\notin$ 2D. Hence, unlike Seiferas' witness, which was proposed based only on intuition, theirs was guaranteed to confirm the conjecture iff the conjecture was true.

Still, Seiferas' intuitively suggested candidate turned out to be 1N-complete, as well [24].[5] We already saw why it is in 1N (Alg. 6), so let us also see why it is 1N-hard. For this, it is enough to homomorphically reduce ONE-WAY LIVENESS to SEPARABILITY. This means (see [24] for the formal details) to provide a systematic way g of replacing each symbol a from the alphabet of ONE-WAY LIVENESS $_h$ and the endmarkers $\vdash, \dashv$ with a string $g(a)$ over the alphabet of SEPARABILITY$_{q(h)}$ so that, for each instance $w = a_1 \cdots a_l$ of ONE-WAY LIVENESS $_h$, performing all replacements preserves membership across the problems:

$$w \in \text{ONE-WAY LIVENESS}_h \iff g(\vdash)g(a_1)\cdots g(a_l)g(\dashv) \in \text{SEPARABILITY}_{q(h)}\,;$$

here, q must be a polynomial. What g should we use? Here is an idea.

> Consider any multi-level graph (Fig. 2c). Imagine "flattening" it by toppling each column to the left, so that its topmost node becomes leftmost (Fig. 2d). Then, an arrow from the ith node of a column to the jth node of the next column spans $2 + (h - i) + (j - 1)$ nodes: its source, its target, all nodes below/after its source, and all nodes above/before its target. We add to this graph two extra nodes, one on the left, pointing at all nodes of the leftmost column, and one on the right, pointed at by all nodes of the rightmost column.
>
> Clearly, the resulting graph contains a path connecting the two extra nodes iff the original graph contains a live path. Moreover, every such path naturally

separates the nodes into groups: each arrow in the path defines the group of all nodes spanned by it, except for its target (Fig. 2d).

We are now ready to produce an instance of separability. We replace each node u with a set of numbers that describe the arrows departing from u. Each arrow is described by the number of nodes between u and its target. E.g., in Fig. 2d, the bold arrow out of the left extra node is described by 2, the extra node itself is replaced by $\{0,1,2,3,4\}$, and the full graph gives rise to the instance

$$\{0,1,\mathbf{2},3,4\}\emptyset\{4\}\{\mathbf{4}\}\{4\}\{4\}\{4,6\}\emptyset\{\mathbf{3},6\}\emptyset\{2,4\}\{5,7\}\{\mathbf{7}\}\emptyset\{4\}\emptyset\{4\}\{3\}\{2\}\{1\}\{\mathbf{0}\}\,.$$

(No set describes the right extra node.) Notice the numbers in bold: they represent the bold arrows of Fig. 2d; and they separate the sets into blocks, in the same way that the bold arrows separate the nodes into groups!

Formally, for each symbol/graph a, we define $g(a) := \alpha_1\alpha_2\cdots\alpha_h$ where

$$\alpha_i := \left\{ (h-i)+(j-1) \;\middle|\; \begin{array}{l} a \text{ contains an arrow from the } i\text{th node of the} \\ \text{left column to the } j\text{th node of the right column} \end{array} \right\};$$

we also set $g(\vdash) := \{0,1,\ldots,h-1\}$ and $g(\dashv) := \{h-1\}\cdots\{1\}\{0\}$. All numbers involved are from 0 to $2h-2$, so the result is an instance of $\text{SEPARABILITY}_{2h-1}$, so $q(h) = 2h-1$. A careful proof can easily be extracted from these observations.

Most attempts to prove 2D $\neq$ 2N have actually focused on confirming this stronger conjecture. The lower bounds mentioned in the introductory section for TWO-WAY LIVENESS [14,28,25,12] were actually proved for ONE-WAY LIVENESS.

A pause. The first goal of this talk so far has been to acquaint the reader with the study of the size complexity of two-way finite automata: the central open question in the area, a motivation behind it, some history, some terminology. Has this goal been achieved for you? Test yourself by answering the following questions: What is 1D? How does it compare to 1N? How does it relate to SEPARABILITY? How does it relate to TWO-WAY LIVENESS? [6] (for the answers)

The second goal of this talk so far has been to convince with examples that the important word in its title is not "Automata", but "Complexity". Automata theory is strongly associated with *computability questions* ("Can such-and-such an automaton recognize language such-and-such?") and with *formal language theory*. In contrast, this talk examines *complexity questions* ("I know such-and-such an automaton can solve problem such-and-such, but how efficient can it be?") and is closer to *computational complexity theory*: we discuss "algorithms" (that happen to run on automata) and "problems" and "reductions" and "complexity classes"and "completeness". Furhtermore, we don't necessarily care about automata or size; we just use this model and resource in the hope of improving our understanding of the properties of general computation.

At this point we are also ready to address a possible complaint: How come we call this "size complexity"? Since we are counting states, shouldn't we call it "state complexity"? The best measure of the size of an automaton, goes the complaint, is the number of bits needed to write down its transition function.

For a 2NFA with σ input alphabet symbols and s states, this is $2\sigma s^2$ bits.[7] So, whenever the symbols greatly outnumber the states, "size" (as number of bits) and "number of states" are hugely different. E.g., the 2NFA implementing Alg. 2 has $2h$ states, but its size is $2 \cdot 2^{(2h)^2} \cdot (2h)^2 = 2^{\Theta(h^2)}$, already close to that of the best known equivalent 2DFA; so, 2D v 2N is about "number of states", not "size"!

This argument is misleading. To see why, let BINARY TWO-WAY LIVENESS be defined over the binary alphabet and differ from TWO-WAY LIVENESS only in that each two-column graph is now encoded in $(2h)^2$ bits. The new problem is still in 2N, again by Alg. 2—it's just that the implementation will now need $O(h^4)$ states, so as to locate the start/end of each graph and the bit representing each arrow, by counting. The new problem is also 2N-complete, by a homomorphic reduction from TWO-WAY LIVENESS—just replace each graph with its binary encoding. So, 2D v 2N is also equivalent to questions of constant alphabet, where "number of states" and "size" are polynomially related. There, removing nondeterminism causes a super-polynomial blow-up either in both measures or in neither, making it safe to use either one. In short, large alphabets are "abbreviations" that allow us to focus on the combinatorial core of a problem. They can always be replaced by small alphabets, where "number of states" and "size" are interchangeable.

So, our use of the term "size complexity" is not wrong. Moreover, it seems advantageous to prefer this term whenever our question about the least upper bound for the blow-up in the number of states is only *whether it is polynomial or not*. The term "state complexity" may then be reserved for the finer part of our studies where, after having answered the polynomiality question, we go on to find the *asymptotic behavior* of the bound or, for even greater detail, its *exact value*; then, "number of states" may behave differently from "size" (as number of bits in description) and/or other measures (e.g., "number of transitions").

A theory to develop. In sharp contrast with TM time/space complexity, where a plethora of complexity classes have been introduced and studied since the 70's [21,29], the study of 2FA size complexity has been progressing very slowly and has stayed focused mostly on 2D v 2N. Figure 3a sketches a map of some TM time complexity classes for three primary modes of computation: *determinism*, *alternation*, and *randomization*.[8] Figure 3b shows what the analogous map should be for 2FAs and size. The key to the analogy is that the time bound $f(n)$ for TMs (where n is the input length) becomes the size bound $f(h)$ for 2FAs (where h is the family index). More specifically, if $\mathcal{X}$ is a mode of computation and $\mathcal{F}$ is a class of functions, then the TM time complexity class

$$\left\{ L \;\middle|\; \begin{array}{l} \text{there exist } \mathcal{X}\text{TM } M \text{ and } f \in \mathcal{F} \text{ such that } M \text{ solves } L \text{ using} \\ \text{at most } f(n) \text{ steps, for all } n \text{ and all } n\text{-long positive instances} \end{array} \right\} \tag{7}$$

corresponds to the 2FA size complexity class

$$\left\{ (L_h)_{h\geq 1} \;\middle|\; \begin{array}{l} \text{there exist } 2\mathcal{X}\text{FAs } (M_h)_{h\geq 1} \text{ and } f \in \mathcal{F} \text{ such that} \\ M_h \text{ solves } L_h \text{ using at most } f(h) \text{ states, for all } h \end{array} \right\}. \tag{8}$$

Let's explore the similarities and differences between these two maps.

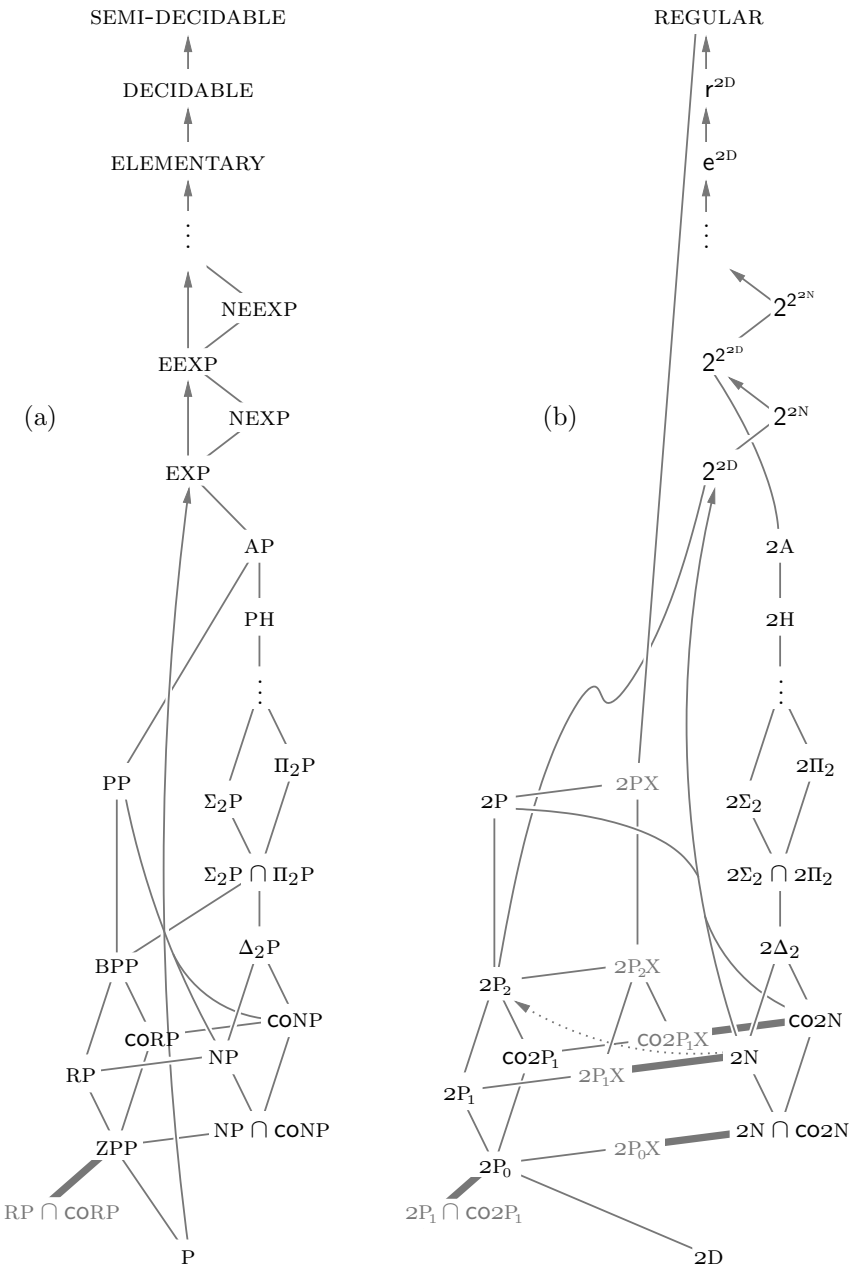


Fig. 3. (a) A map of some TM time complexity classes. (b) The corresponding 2FA size complexity map. A bold line $\mathcal{C}$–$\mathcal{C}'$ means $\mathcal{C} = \mathcal{C}'$; a simple line from $\mathcal{C}$ upwards to $\mathcal{C}'$ means $\mathcal{C} \subseteq \mathcal{C}'$; an arrow $\mathcal{C} \longrightarrow \mathcal{C}'$ means $\mathcal{C} \subsetneq \mathcal{C}'$; a dotted arrow $\mathcal{C} \dashrightarrow \mathcal{C}'$ means $\mathcal{C} \not\supseteq \mathcal{C}'$.

Determinism. If $\mathcal{X}$ is determinism and $\mathcal{F}$ is all *polynomial* functions, then (7) and (8) define P and 2D, respectively. If $\mathcal{F}$ is all *exponential* functions ($2^{p(n)}$, for polynomial p), then DTMs define EXP, while 2DFAs define the class of problem families that are solvable with at most exponentially many states—we propose the name $2^{2^{\mathrm{D}}}$. Similarly, if $\mathcal{F}$ is all *doubly-exponential* functions, we get EEXP and $2^{2^{2^{\mathrm{D}}}}$ (again, a proposed name); and so on, for higher exponentials. When $\mathcal{F}$ becomes all *elementary* functions, we get the union of all these classes on each side; for DTMs, this is ELEMENTARY; for 2DFAs, we propose the name $\mathsf{e}^{2^{\mathrm{D}}}$. Further up, decidable problems is the TM class when $\mathcal{F}$ is all *recursive* functions; for the corresponding 2FA class, we propose the name $\mathsf{r}^{2^{\mathrm{D}}}$. Finally, if $\mathcal{F}$ is *all* functions, we get all semi-decidable problems and all families of regular problems.

The deterministic size complexity classes are all closed under complement—below the elementary bounds, this is non-trivial [27,9]. In addition, the well-known strict hierarchy

$$\mathrm{P} \subsetneq \mathrm{EXP} \subsetneq \mathrm{EEXP} \subsetneq \cdots \subsetneq \mathrm{ELEMENTARY} \subsetneq \mathrm{DECIDABLE} \subsetneq \mathrm{SEMI\text{-}DECIDABLE}$$

maps to a hierarchy that is also strict:

$$2\mathrm{D} \subsetneq 2^{2^{\mathrm{D}}} \subsetneq 2^{2^{2^{\mathrm{D}}}} \subsetneq \cdots \subsetneq \mathsf{e}^{2^{\mathrm{D}}} \subsetneq \mathsf{r}^{2^{\mathrm{D}}} \subsetneq \mathrm{REGULAR}.$$

Inclusions are trivial.[9] Strictness follows from the fact that the minimum number of states on a 2DFA solving the unary singleton problem $\{0^x\}$ is $x+1$ [2].[10] So, for an appropriately selected f, the family $(\{0^{f(h)}\})_{h\geq 1}$ can witness any of the above differences; e.g., to show $\mathsf{e}^{2^{\mathrm{D}}} \subsetneq \mathsf{r}^{2^{\mathrm{D}}}$, just let f be recursive but non-elementary.

Alternation. If $\mathcal{X}$ is alternation and $\mathcal{F}$ is all polynomial functions, then we arrive at alternating TMs (ATMs) of polynomial time and the class AP. In studying this class, people have distinguished subclasses of problems by restricting the maximum number of runs of existential and universal steps that the ATM may perform throughout a computation. Fixing this number to a particular $i \geq 0$, we get the two classes $\Sigma_i\mathrm{P}$ and $\Pi_i\mathrm{P}$, depending on whether the first run consists of existential or universal steps, respectively. This way, $\mathrm{P} = \Sigma_0\mathrm{P} = \Pi_0\mathrm{P}$, $\mathrm{NP} = \Sigma_1\mathrm{P}$, $\mathsf{co}\mathrm{NP} = \Pi_1\mathrm{P}$, and the infinite *polynomial-time hierarchy* rises above them, which may or may not be strict. Equivalently, one can think of $\Sigma_i\mathrm{P}$ as the problems that are solvable in polynomial time by a NTM with access to an oracle for a problem in $\Sigma_{i-1}\mathrm{P}$, namely as $\mathrm{NP}^{\Sigma_{i-1}\mathrm{P}}$; and of $\Pi_i\mathrm{P}$ as all their complements, namely as $\mathsf{co}\mathrm{NP}^{\Sigma_{i-1}\mathrm{P}}$. Then, the class $\Delta_i\mathrm{P}$ can also be considered, defined analogously but with a DTM, namely as $\mathrm{P}^{\Sigma_{i-1}\mathrm{P}}$. By the definitions and a few relatively easy observations, we end up with the well-known relationships:

$$\mathrm{P} \subseteq \mathrm{NP} \cap \mathsf{co}\mathrm{NP} \begin{array}{c} \subsetneq \\ \subsetneq \end{array} \begin{array}{c} \mathrm{NP} \\ \mathsf{co}\mathrm{NP} \end{array} \begin{array}{c} \subsetneq \\ \subsetneq \end{array} \Delta_2\mathrm{P} \subseteq \Sigma_2\mathrm{P} \cap \Pi_2\mathrm{P} \begin{array}{c} \subsetneq \\ \subsetneq \end{array} \begin{array}{c} \Sigma_2\mathrm{P} \\ \Pi_2\mathrm{P} \end{array} \begin{array}{c} \subsetneq \\ \subsetneq \end{array} \cdots \subseteq \mathrm{PH} \subseteq \mathrm{AP} \subseteq \mathrm{EXP} \qquad (9)$$

where PH is the union of all restricted classes.

Analogous complexity classes can be considered for 2FAs and size—we propose[11] the names $2\Sigma_i$ and $2\Pi_i$ for the ith level of the hierarchy, 2H for the union

of all levels, and 2A for all problem families solvable by alternating 2FAs (2AFAs) with polynomially many states. E.g., a problem family should be in $2\Sigma_i$ iff its hth member can be solved by a small ($p(h)$-state, for some polynomial p) 2AFA that performs $\leq i$ runs of existential and universal steps per computation, starting with existential ones. This way, $2\mathrm{D} = 2\Sigma_0 = 2\Pi_0$, $2\mathrm{N} = 2\Sigma_1$, and $\mathsf{co2N} = 2\Pi_1$.

It should also be possible to work with the oracle-based definition. E.g., a problem family should be in the class $2\Delta_2 = 2\mathrm{D}^{2\mathrm{N}}$ if its hth member can be solved by a small 2DFA that has access to an oracle which responds to any question that can be answered by a small 2NFA executed on the same input. This way, the join TWO-WAY LIVENESS $\bowtie$ $\overline{\text{TWO-WAY LIVENESS}}$, defined as[12]

> Given an instance w of TWO-WAY LIVENESS check that
> *either* w has even length and is live *or* it has odd length and is dead.

is in $2\Delta_2$ by the straightforward algorithm

> We scan the input once to check whether its length is even. We return to the leftmost symbol and call the oracle to check whether the input is live. If the two checks returned the same result, we accept; otherwise, we reject.

but is not known to be in $2\mathrm{N} \cup \mathsf{co2N}$ (if it were, then we could disprove[13] the conjecture $2\mathrm{N} \neq \mathsf{co2N}$). Similarly, one can define $2\mathrm{N}^{2\mathrm{N}}$, $\mathsf{co2N}^{2\mathrm{N}}$, etc. However, some work is necessary in order to clarify these definitions: one should describe how exactly oracle calls work[14] and compare with the earlier definitions (is $2\Sigma_i = 2\mathrm{N}^{2\Sigma_{i-1}}$?). Such work is beyond the purposes of this exploratory exposition.

In the end, after appropriate fine-tuning, we should probably be able to produce a situation similar to the one in (9):

$$2\mathrm{D} \subseteq 2\mathrm{N} \cap \mathsf{co2N} \subsetneqq \begin{matrix} 2\mathrm{N} \\ \mathsf{co2N} \end{matrix} \subsetneqq 2\Delta_2 \subseteq 2\Sigma_2 \cap 2\Pi_2 \subsetneqq \begin{matrix} 2\Sigma_2 \\ 2\Pi_2 \end{matrix} \subsetneqq \cdots \subseteq 2\mathrm{H} \subseteq 2\mathrm{A} \subseteq 2^{2^{2\mathrm{D}}}$$

Note that, for a tight analogy with (9), the last inclusion should be AP $\subseteq 2^{2^{2\mathrm{D}}}$. But this seems not to known. The listed inclusion follows from [18].[15]

Randomization. If $\mathcal{X}$ is randomization, we need to clarify what it means for a probabilistic TM (PTM) or probabilistic 2FA (2PFA) M to solve a problem L. Depending on what we want to model, this can be done in different ways:

two-sided error: To model all nontrivial probabilistic algorithms, we require that a *cut-point* $\lambda \in (0, 1)$ distinguishes between positive and negative instances: each $w \in L$ is accepted w.p. $> \lambda$ and each $w \notin L$ is accepted w.p. $< \lambda$. Then, the *completeness* $c(n)$ of M is the smallest acceptance probability over positive n-long instances; and the *soundness* $s(n)$ of M is the largest acceptance probability over negative n-long instances. Hence, the cut-point separates completeness and soundness: $s(n) < \lambda < c(n)$, for all n. The difference $c(n) - s(n)$ is the *isolation* of the cut-point.

two-sided error of bounded probability: To model all practical probabilistic algorithms (i.e., those allowing us to efficiently extract statistically reliable answers by sampling and majority vote), we further require that the isolation

of the cut-point is significant in the length of the input: $c(n) - s(n) \geq \frac{1}{r(n)}$, for some polynomial r and all n.

one-sided error of bounded probability: To model all Monte Carlo algorithms, we further require that M never errs on negative instances, namely $s(n) = 0$.

(*"zero-sided"*) *error of zero probability*: To model all Las Vegas algorithms, we require that M always halts with either the correct answer or no answer at all and that, for all n and all n-long instances, the probability of it returning an answer is significant ($\geq \frac{1}{r(n)}$, for some polynomial r).

The TM time complexity classes that correspond to these requirements when $\mathcal{F}$ is all *polynomial* functions are PP, BPP, RP, and ZPP, respectively. For the 2FA size complexity analogues, we propose[16] the names 2P, $2P_2$, $2P_1$, and $2P_0$—and call the corresponding automata 2PFAs, $2P_2$FAs, $2P_1$FAs, and $2P_0$FAs.

Still, some further clarifications are necessary.

I. *Isolation*: In the bounded-error models, we require that the cut-point isolation be significant in the input length ($\geq \frac{1}{r(n)}$ for some polynomial r). This way, given the probabilistic machine and an n-long input w, one can extract from the machine a statistically reliable answer about w efficiently (in time polynomial in n). This is true irrespective of whether the machine is a standalone PTM M or a member M_h of a $2P_2$FA-family. In the latter case, however, we must require that the isolation be significant in h as well (or else we may lose the connections to TM space complexity—via theorems à la Berman and Lingas [1]). Hence, for $2P_2$ and $2P_1$ we require that the cut-point isolation of M_h on n-long instances is $\geq \frac{1}{r(h,n)}$, for some polynomial r and all h, n. Similarly, for $2P_0$ we require that M_h returns an answer w.p. $\geq \frac{1}{r(h,n)}$.

II. *Time complexity*: In contrast to deterministic and alternating 2FAs, where accepting computations are always at most linearly longer than the input, a probabilistic 2FA may very well run much slower: when finite, its expected running time may be exponential in the input length. Hence, to describe *efficient* computation, our complexity classes must also require that the expected time is polynomial in n—and also in h, for reasons similar as above. E.g., $2P_2$ must be

$$\left\{ (L_h)_{h\geq 1} \;\middle|\; \begin{array}{l} \text{there exist } 2P_2\text{FAs } (M_h)_{h\geq 1} \text{ and polynomials } p, q \text{ such that} \\ M_h \text{ solves } L_h \text{ using at most } p(h) \text{ states and } q(h,n) \text{ steps} \\ \text{on average, for all } h \text{ and all } n \text{ and all } n\text{-long instances} \end{array} \right\},$$

and similarly for the other classes. Still, the case of polynomial size but exponential expected time is not uninteresting. To discuss such "small but slow" algorithms, we propose the names 2PX, $2P_2$X, $2P_1$X, and $2P_0$X, respectively.

III. *Fineness of distributions*: A probabilistic 2FA can be *coin-flipping*, if the probability of each transition is either 0 or $\frac{1}{2}$ or 1; or *rational*, if rational transition probabilities are allowed; or *real*, if real transition probabilities are allowed. To describe *discrete* efficient computation, we must assume that our complexity classes have been defined based on automata of the first kind. Still, one can prove that every rational 2FA has an equivalent coin-flipping 2FA that is at most linearly larger and slower. Hence, redefining our classes on the basis of rational

automata would not affect them. Finally, to discuss the variant classes that we get when we let all 2FAs be real, we propose the names real-$2P_0$, real-$2P_1X$, etc.

IV. *Regularity*: It is easy to see that $2P_0$FAs and $2P_1$FAs can solve only regular problems. In contrast, $2P_2$FAs can solve only regular problems iff we restrict their expected time to be polynomial [5,7], and 2PFAs can solve non-regular problems even with polynomial expected time [5]. Hence, in order to keep all members of every family in our classes regular, the definitions of $2P_2X$, 2P, 2PX must include the explicit requirement that "each L_h is regular".

With these clarifications, we are ready to list some known facts. First of all, the well-known relationships

$$\mathrm{P} \subseteq \mathrm{ZPP} = \mathrm{RP} \cap \mathsf{co}\mathrm{RP} \subseteq \mathrm{RP} \subseteq \mathrm{BPP} \subseteq \mathrm{PP}$$

translate directly (by the definitions and an easy fact) to the relationships

$$2\mathrm{D} \subseteq 2\mathrm{P}_0 = 2\mathrm{P}_1 \cap \mathsf{co}2\mathrm{P}_1 \subseteq 2\mathrm{P}_1 \subseteq 2\mathrm{P}_2 \subseteq 2\mathrm{P},$$

and similarly for the $*$X classes; also, we clearly have $2P_0 \subseteq 2P_0X$, $2P_1 \subseteq 2P_1X$, etc. Moreover, it can be proved (using the ideas of [19]) that the freedom to be slow allows Monte Carlo and Las Vegas automata to simulate nondeterminism:

$$2\mathrm{P}_1\mathrm{X} = 2\mathrm{N} \qquad \text{and thus} \qquad 2\mathrm{P}_0\mathrm{X} = 2\mathrm{P}_1\mathrm{X} \cap \mathsf{co}2\mathrm{P}_1\mathrm{X} = 2\mathrm{N} \cap \mathsf{co}2\mathrm{N}.$$

Finally, we also know (by the theorems of [5, Sect. 6]) that small & fast $2P_2$FAs can be simulated by large 2DFAs, but not by small ones:[17]

$$2\mathrm{D} \subsetneq 2\mathrm{P}_2 \subseteq 2^{2^{2\mathrm{D}}}$$

but small & slow $2P_2$FAs may even need non-recursively larger 2DFA simulators:

$$2\mathrm{P}_2\mathrm{X} \not\subseteq \mathsf{r}^{2^{2\mathrm{D}}},$$

i.e., no recursive function can upper bound the size of the simulating 2DFAs.

Programmatic access. Although some of the open questions posed by the diagram of Fig. 3b are certainly hard, none seems to be hopeless. Moreover, each of them can be approached via three other questions of gradually decreasing difficulty: the corresponding questions for *sweeping*, *rotating*, and *one-way* automata (Fig. 4). A 2FA is sweeping (SFA: SDFA, SNFA, etc.) if its head can turn only on the end-markers, so that each computation is a series of one-way scans of alternate directions; it is rotating (RFA: RDFA, RNFA, etc.) if its head can only move right or jump from the right end-marker to the left one, so that each computation is a series of rightward scans; and it is one-way (1FA: 1DFA, 1NFA, etc.) if its head moves always right, in a single rightward scan.

So, e.g., if the full 2D v 2N problem seems hard, we can step back and study the relationship between determinism and nondetermism for SFAs first: Can SDFAs

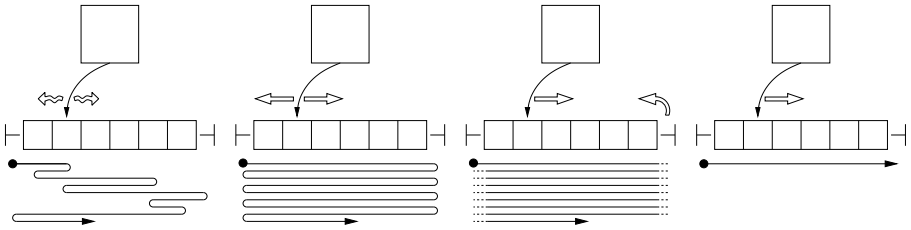


Fig. 4. The full range: *two-way*, *sweeping*, *rotating*, and *one-way* automata.

always stay at most polynomially larger than SNFAs? Or, introducing the classes SD and SN (as in (4) but for SDFAs and SNFAs), we can ask the restriction:

$$\mathrm{SD} = \mathrm{SN}\ ?$$

If this is still hard, we can attack the even simpler question for RDFAs and RNFAs:

$$\mathrm{RD} = \mathrm{RN}\ ?$$

for RD and RN defined analogously. Finally, our last retreat is the one-way case:

$$1\mathrm{D} = 1\mathrm{N}\ ?$$

These same simplifying steps can be made in the study of any relationship in Fig. 3b. Typically, solving the one-way case is indeed a lot easier than all other cases. Then, a serious boost of ideas is required for the rotating case; here, an indispensible lower-bound technique is Sipser's "generic strings" method [28], in which one studies the behavior of the automaton on inputs that are long enough to minimize a carefully chosen measure.[18] The sweeping case is then relatively easy; one just needs to carefully exploit symmetry. Finally, moving from the sweeping to the two-way case is currently beyond our reach, in general.

Another natural restriction one can focus on is that of *unary* automata. For each class $\mathcal{C}$ in Fig. 3b, one can consider the class $\mathsf{unary}\text{-}\mathcal{C}$ that is defined identically to $\mathcal{C}$ but for unary automata. For example, one can ask:

$$\mathsf{unary}\text{-}2\mathrm{D} = \mathsf{unary}\text{-}2\mathrm{N}\ ?$$

Although a lot simpler, the unary case can still be highly demanding. Moving from it to the multi-symbol case is currently again beyond our reach, in general.

Some facts. When our questions are asked for the restricted models, as opposed to full-fledged 2FAs, the diagram of Fig. 3b changes as in Fig. 5. More specifically:

For SFAs (Fig. 5s), we have confirmed that nondeterminism beats determinism: $\mathrm{SD} \subsetneq \mathrm{SN}$ [28],[19] which directly implies $2^{\mathrm{SD}} \subsetneq 2^{\mathrm{SN}}$, $2^{2^{\mathrm{SD}}} \subsetneq 2^{2^{\mathrm{SN}}}$, etc. In fact, we even know that in the series of trivial inclusions

$$\mathrm{SD} \overset{1}{\subseteq} \mathrm{SP}_0 \overset{2}{\subseteq} \mathrm{SP}_0\mathrm{X} = \mathrm{SP}_1\mathrm{X} \cap \mathsf{co}\mathrm{SP}_1\mathrm{X} = \mathrm{SN} \cap \mathsf{co}\mathrm{SN} \overset{3}{\subseteq} \mathrm{SN}\,,$$

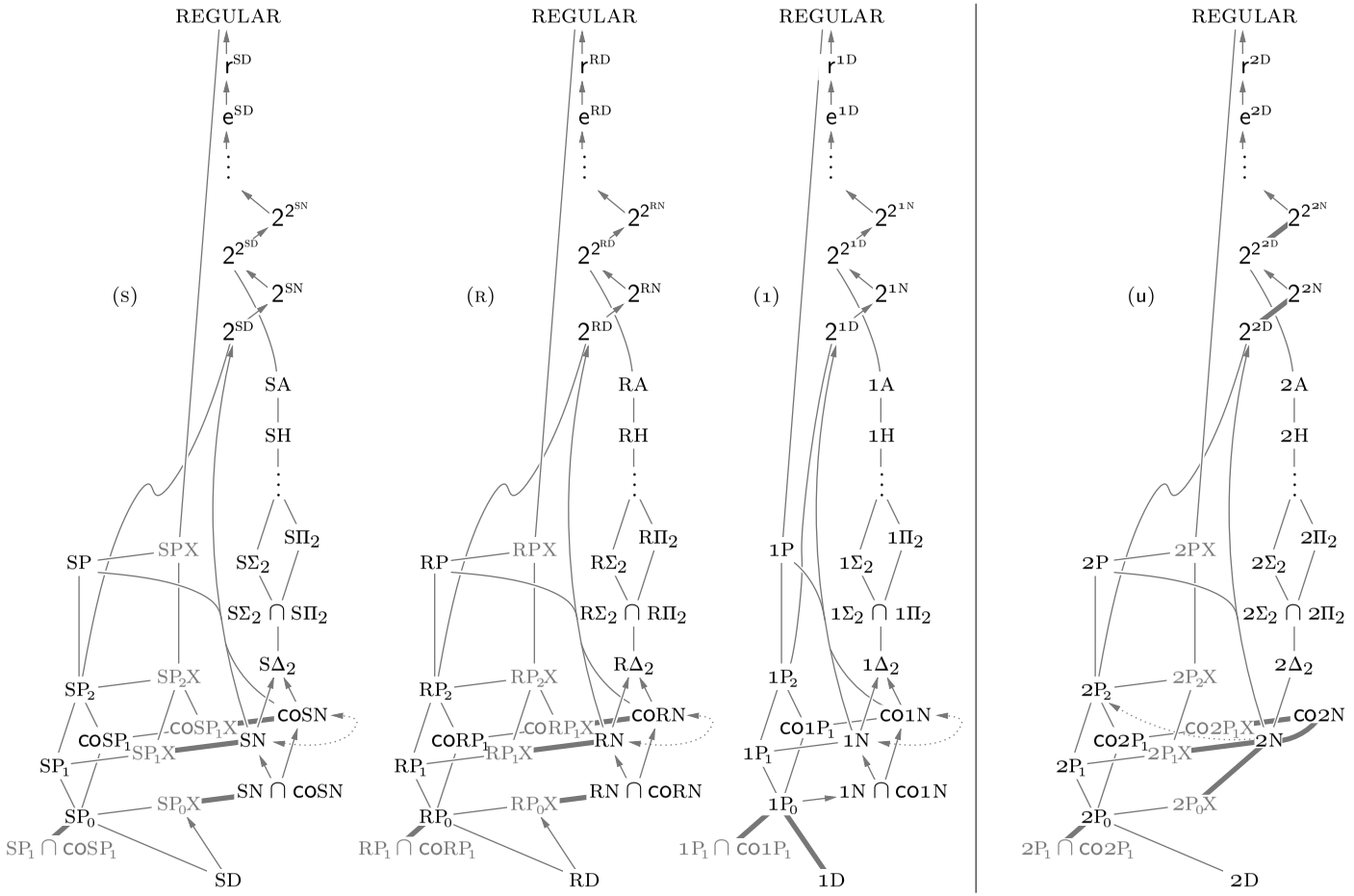


Fig. 5. The diagrams derived from that of Fig. 3b when all questions are asked not for full-fledged 2FAs, but (S) for SFAs; (R) for RFAs; (1) for 1FAs; (u) for unary 2FAs (for simplicity, we omit the unary- prefixes). The meaning of lines and arrows is as in Fig. 3.

both 3 and at least one of $1, 2$ are strict [15,13]. That one of $1, 2$ is strict follows from the fact that $\text{SD} \neq \text{SP}_0\text{X}$—i.e., slow Las Vegas behavior beats determinism [15].[20] Note that, although we can confirm the strictness of neither inclusion, we do know that 2 is strict in the special case where the fast SP_0FAs must run in linear expected time (as opposed to arbitrary polynomial) [17]. That inclusion 3 is strict follows from the fact that $\text{SN} \neq \text{co}\text{SN}$—i.e., nondeterminism is not closed under complement [13].[21] Note that this easily implies that $\text{SN} \cup \text{co}\text{SN} \subsetneq \text{S}\Delta_2$, as well.[22] The remaining relationships in Fig. 5S hold for the same reasons as for 2FAs. Note that there is no arrow to indicate $\text{SP}_2 \not\subseteq \text{SN}$, as the witness of [5, Thm 6.2.1] for $\text{2P}_2 \not\subseteq \text{2N}$ needs the full bidirectionality of the 2P_2FA.

The diagram for RFAs (Fig. 5R) is identical to that for SFAs, for essentially the same reasons. Typically, a theorem for the sweeping case comes with a proof that is stronger than the statement (as indicated in the Notes) and, in fact, implies the theorem for the rotating case. In addition, sometimes small RFAs already have all the power of small SFAs (e.g., $\text{RN} = \text{SN}$, $\text{RP}_2\text{X} = \text{SP}_2\text{X}$, $\text{RPX} = \text{SPX}$ [15]), and thus a theorem for either case implies the same for the other one.

The diagram for 1FAs (Fig. 51) is not very different. Once again, we know that each one of the trivial inclusions $\text{1D} \subseteq \text{1N} \cap \text{co1N} \subseteq \text{1N}, \text{co1N} \subseteq \text{1}\Delta_2$ is strict [24, §4.1].[23] But now, of course, there are no probabilistic classes for exponential expected time. In addition, we know that $\text{1D} = \text{1P}_0$—i.e., Las Vegas behavior is no more powerful than determinism [11].

Finally, for unary 2FAs (Fig. 5u) important differences exist. First, a subexponential upper bound is known for the increase in size when removing nondeterminism [8]. So, starting at exponential size, nondeterminism is not essential:[24]

$$\text{unary-2}^{2^{\text{2D}}} = \text{unary-2}^{2^{\text{2N}}}, \quad \text{unary-2}^{2^{2^{\text{2D}}}} = \text{unary-2}^{2^{2^{\text{2N}}}}, \quad \text{etc.}$$

Second, nondeterminism is closed under complement [9]:

$$\text{unary-2N} = \text{unary-co2N}\,,$$

which implies that slow Las Vegas behavior is as powerful as nondeterminism:

$$\text{unary-2P}_0\text{X} = \text{unary-2P}_1\text{X} \cap \text{unary-co2P}_1\text{X} = \text{unary-2N} \cap \text{unary-co2N} = \text{unary-2N}\,,$$

Overall, the evidence in the unary case is that nondeterminism offers no significant advantage over determinism, which contrasts with what we know for the one-way, rotating, and sweeping cases and what we believe for the two-way case.

Conclusion. This has been a semi-formal talk on the size complexity of two-way finite automata. In the first half, we presented a central open problem and the main concepts in the area, explained a motivation, and recalled some early history. In the second half, we sketched where the area is heading for, if it is to mimic the development of Turing machine time/space complexity. We expressed all our statements in terms of size complexity classes (rather than the commoner "trade-off" vocabulary) and proposed names where necessary—all in continuation and in the style of the Sakoda-Sipser framework [24]. We then

described how each open question may be approached via restrictions to the unary alphabet or to the sweeping, rotating, or one-way input head. Finally, we expressed in this framework some of the progress that has been achieved so far. Our exposition has tried to be welcoming and informative, rather than rigorous or complete, and it represents this author's perspective on the subject.

The diagram of Fig. 3b remains, for the most part, unexplored: an open question lies behind any line that is not an arrow, and behind any pair of classes with no upward path between them. To a lesser but still great extent, the same is true of the diagrams of Fig. 5. A few of these questions may have already been answered—in which case this author offers his apologies for not knowing/realizing it. Other questions will be relatively easy, especially in cases where the corresponding question for TM complexity has been answered. Still, the (many) remaining questions will be hard, although certainly not impossible.

In studying these questions one will probably need to choose appropriate definitions where necessary (e.g., for oracle-2FAs), identify new complete problems (e.g., for $2\Sigma_i$, 2A), introduce new types of reductions (e.g., more powerful than the homomorphic ones), explore connections with time/space complexity (e.g., by extending the Berman-Lingas theorem [1]), add other modes of computation into the picture (e.g., interaction, the quantum mode), and more.

Much like the questions themselves, some of the ideas for answering them may come directly from answers that have already been given to corresponding questions in TM time/space complexity (e.g., inductive counting was borrowed from the proof of NL = coNL to help prove unary-2N = unary-co2N [9]). By testing these ideas in new settings, we can explore their limits and deepen our understanding of their power (e.g., inductive counting appears inadequate for showing 2N = co2N; and it will eventually prove so, if the conjecture 2N $\neq$ co2N is true). In turn, this may help us arrive at extensions or completely new techniques, hopefully advancing our understanding of TM complexity as well.

References

1. 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.
2. J.-C. Birget. Two-way automata and length-preserving homomorphisms. Report 109, Department of Computer Science, University of Nebraska, 1990.
3. J.-C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical Systems Theory*, 26:237–269, 1993.
4. M. Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.
5. C. Dwork and L. J. Stockmeyer. A time complexity gap for two-way probabilistic finite-state automata. *SIAM Journal of Computing*, 19(6):1011–1023, 1990.
6. C. Dwork and L. J. Stockmeyer. Finite state verifiers I: The power of interaction. *Journal of the ACM*, 39(4):800–828, 1992.
7. R. Freivalds. Probabilistic two-way machines. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 33–45, 1981.

8. V. Geffert, C. Mereghetti, and G. Pighizzini. Converting two-way nondeterministic unary automata into simpler automata. *Theoretical Computer Science*, 295:189–203, 2003.
9. V. Geffert, C. Mereghetti, and G. Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007.
10. J. Goldstine, M. Kappes, C. M. R. Kintala, H. Leung, A. Malcher, and D. Wotschke. Descriptional complexity of machines with limited resources. *Journal of Universal Computer Science*, 8(2):193–234, 2002.
11. J. Hromkovič and G. Schnitger. On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. *Information and Computation*, 169:284–296, 2001.
12. J. Hromkovič and G. Schnitger. Nondeterminism versus determinism for two-way finite automata: generalizations of Sipser's separation. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, pages 439–451, 2003.
13. C. Kapoutsis. Small sweeping 2NFAs are not closed under complement. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, pages 144–156, 2006.
14. C. Kapoutsis. Deterministic moles cannot solve liveness. *Journal of Automata, Languages and Combinatorics*, 12(1-2):215–235, 2007.
15. C. Kapoutsis, R. Královič, and T. Mömke. An exponential gap between Las Vegas and deterministic sweeping finite automata. In *Proceedings of the International Symposium on Stochastic Algorithms: Foundations and Applications*, pages 130–141, 2007.
16. C. Kapoutsis, R. Královič, and T. Mömke. On the size complexity of rotating and sweeping automata. In *Proceedings of the International Conference on Developments in Language Theory*, pages 455–466, 2008.
17. R. Královič. Infinite vs. finite space-bounded randomized computations. In *Proceedings of the IEEE Conference on Computational Complexity*, 2009. To appear.
18. R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal of Computing*, 13(1):135–155, 1984.
19. I. I. Macarie and J. I. Seiferas. Amplification of slight probabilistic advantage at absolutely no cost in space. *Information Processing Letters*, 72(3–4):113–118, 1999.
20. A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *Proceedings of the Symposium on Switching and Automata Theory*, pages 188–191, 1971.
21. C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, MA, 1994.
22. M. O. Rabin. Two-way finite automata. In *Proceedings of the Summer Institute of Symbolic Logic*, pages 366–369, Cornell, 1957.
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 the Symposium on the Theory of Computing*, pages 275–286, 1978.
25. J. I. Seiferas. Untitled manuscript. Communicated to M. Sipser, Oct. 1973.
26. J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3:198–200, 1959.
27. M. Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10:335–338, 1980.

28. M. Sipser. Lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences*, 21(2):195–202, 1980.
29. M. Sipser. *Introduction to the theory of computation.* PWS Publishing Company, Boston, MA, 1996.

Notes

[1]Assuming that the observable universe contains 10^{80} atoms, the sun runs out of fuel in 10 billion years, and drawing 1 state takes 1 atom and 1 picosecond.

[2]By now the reader has probably picked up our naming conventions. But, for one last time, let's just make sure: "1NFA" means *one-way nondeterministic finite automaton.*

[3]The "LIVENESS" part of the name hints at the behavior of problem instances under extension: A path from the leftmost to the rightmost column is called *live*; an instance that contains live paths is called *live*, as well; an instance that contains no live paths is called *dead*. Now think of what happens when we prepend or append extra symbols/graphs to an instance: if the instance is live, it may remain live or become dead; in contrast, if the instance is dead, it will remain dead—a most tight analogy. The "TWO-WAY" part hints at the fact that paths may grow in either direction.

[4]In fact, his conjecture was even stronger: that the minimum number of states in a 2DFA solving SEPARABILITY$_h$ is exactly 2^h—as it is for 1DFAs.

[5]One can also prove the same for all other candidate witnesses that he proposed.

[6]By analogy to 2D (and 2N and 1N), 1D is the class of problem families that can be solved by families of 1DFAs of polynomially growing size. The formal definition is as in (4) but for 1DFAs. We have $\text{1D} \subseteq \text{1N}$ (trivially) and $\text{1D} \neq \text{1N}$ (e.g., the problem family implicit in (3) is in $\text{1N} - \text{1D}$). Overall, $\text{1D} \subsetneq \text{1N}$, and thus also $\text{1D} \subsetneq \text{2N}$ (since $\text{1N} \subseteq \text{2N}$). Since SEPARABILITY is 1N-complete, we know SEPARABILITY $\notin$ 1D. Since TWO-WAY LIVENESS is 2N-complete, we know TWO-WAY LIVENESS $\notin$ 1D. (Here we are using the fact that 1D is closed under homomorphic reductions.)

[7]For each direction d (left, right), input symbol a, and pair of states (p, q), we need 1 bit saying whether being at p and reading a causes the automaton to jump to q and move its head in the d direction.

[8]One could also include here one more map for TM space complexity and/or augment all maps with *interaction* [6], *parallelism*, the *quantum* mode, etc. But time complexity and the three primary modes are enough to make our point.

[9]In fact, [26] implies that even $\text{2N} \subseteq 2^{\text{1D}}$, $2^{\text{2N}} \subseteq 2^{2^{\text{1D}}}$, etc.

[10]In fact, $x + 1$ states are sufficient even for a 1DFA; and they are necessary even for a 2NFA [2, Fact 5.2].

[11]Here, we follow the Sakoda-Sipser two-symbol naming convention: one symbol for the head mode, one more for the transition function mode—as in "2D", "1N", etc.

[12]See [24, §4.1] for a similar join, witnessing that $\text{2D} \not\subseteq \text{1N} \cup \text{co1N}$. Also, see Note 3 for what it means for an instance of TWO-WAY LIVENESS to be live/dead.

[13]*Proof*: Suppose the join is in $\text{2N} \cup \text{co2N}$. W.l.o.g., assume it is in 2N (if in co2N, work similarly but with even lengths). Let M be a small 2NFA solving the join. Using M, we can construct a small 2NFA M' solving $\overline{\text{TWO-WAY LIVENESS}}$. Here is how: We scan the input w once to check the parity of its length. If odd, we just simulate M on w—and thus end up accepting iff w is dead. If even, we simulate M on rw, where r is the two-column graph that contains all $(2h)^2$ arrows—since rw is of odd length, we end up acepting iff rw is dead, and thus iff w is dead. It should be clear that M'

can indeed implement this algorithm, and thus solves $\overline{\text{TWO-WAY LIVENESS}}$ with roughly twice as many states as M. This implies $\overline{\text{TWO-WAY LIVENESS}} \in \text{2N}$, and thus $\text{co2N} = \text{2N}$.

[14]How does the oracle read a query? Is the query always the entire input of the 2FA, is it some portion of the input, or is it produced from the input by a small two-way transducer? How does the 2FA read the oracle's answer?

[15]In fact, [18, Thm 4.2.1] proves that even $2^{2^{1D}}$ contains 2A.

[16]Again, we follow the Sakoda-Sipser naming convention: "P" means "probabilistic" and the index counts the sides of bounded error or, if the error is unbounded, is absent.

[17]In fact, [5] proves much more: Thm 6.1 says that even $\text{real-2P}_2 \subseteq 2^{1D}$ (small & fast 2P_2FAs can be simulated by large 2DFAs even when they are *real* and even when the 2DFAs are actually *one-way*) and Thm 6.2 says that even $\text{2N} \not\supseteq \text{2P}_2$ (small 2DFAs cannot simulate every small & fast 2P_2FA even if they are allowed to use nondeterminism).

[18]The method was first applied to deterministic (rotating/sweeping) automata [28], then also to nondeterministic ones [13] and to probabilistic ones [17]. For other applications to deterministic automata, see [14,15,16].

[19]In fact, [28] proves that even $\text{SD} \not\supseteq \text{1N}$.

[20]In fact, [15] proves that even $\text{SD} \not\supseteq \text{1N} \cap \text{co1N}$.

[21]In fact, [13] proves that even $\text{coSN} \not\supseteq \text{1N}$.

[22]In [13], the witness for $\text{SN} \not\subseteq \text{coSN}$ is ONE-WAY LIVENESS. So, consider the join ONE-WAY LIVENESS $\bowtie \overline{\text{ONE-WAY LIVENESS}}$. As in Note 13, we can easily prove that (i) the join is in SD^{SN} and (ii) if it were in $\text{SN} \cup \text{coSN}$, we would have ONE-WAY LIVENESS $\in \text{coSN}$.

[23]For the strictness of $\text{1N} \cup \text{co1N} \subseteq \text{1}\Delta_2$, consider the join T_n used in [24, §4.1].

[24]In fact, [8] implies that even $\text{unary-2}^{\text{SD}} \supseteq \text{unary-2}^{2^{\text{N}}}$, $\text{unary-2}^{2^{\text{SD}}} \supseteq \text{unary-2}^{2^{2^{\text{N}}}}$, etc.