Predicate Characterizations in the Polynomial-Size Hierarchy

Christos A. Kapoutsis

Carnegie Mellon University in Qatar

Abstract. The *polynomial-size hierarchy* is the hierarchy of 'minicomplexity' classes which correspond to *two-way alternating finite automata* with polynomially many states and finitely many alternations. It is defined by analogy to the *polynomial-time hierarchy* of standard complexity theory, and it has recently been shown to be strict above its first level.

It is well-known that, apart from their definition in terms of polynomial-time *alternating Turing machines*, the classes of the polynomial-time hierarchy can also be characterized in terms of polynomial-time *predicates*, polynomial-time *oracle Turing machines*, and *formulas* of second-order logic. It is natural to ask whether analogous alternative characterizations are possible for the polynomial-size hierarchy, as well.

Here, we answer this question affirmatively for predicates. Starting with the first level of the hierarchy, we experiment with several natural ways of defining what a 'polynomial-size predicate' should be, so that existentially quantified predicates of this kind correspond to polynomial-size *two-way nondeterministic finite automata*. After reaching an appropriate definition, we generalize to every level of the hierarchy.

1 Introduction

The k-th level of the *polynomial-size hierarchy* consists of the classes $2\Sigma_k$ and $2\Pi_k$ of all (families of) regular languages which are decided by (families of) *two-way alternating finite automata* (2AFAs) with polynomially many states (i.e., of polynomial 'size'), where the start state is respectively existential or universal and every computation path on any input alternates $< k$ times between existential and universal steps, if $k > 0$; or uses only deterministic steps, if $k = 0$. The question whether this hierarchy is strict was raised in [6] and answered in the affirmative by Geffert [3] for all levels above the lowest two: for all $k \geq 1$,

$$2\Sigma_k \subsetneq 2\Sigma_{k+1} \quad \text{and} \quad 2\Sigma_k \not\subseteq 2\Pi_k \;\;\&\;\; 2\Sigma_k \not\supseteq 2\Pi_k \quad \text{and} \quad 2\Pi_k \subsetneq 2\Pi_{k+1}\,.$$

For $k = 0$, the question is still open: the classes $2\Sigma_0$ and $2\Sigma_1$ are respectively the classes $2\mathsf{D}$ and $2\mathsf{N}$ of all (families of) regular languages decided by (families of) *deterministic* and *nondeterministic two-way finite automata* (2DFAs and 2NFAs) with polynomially many states; hence, proving that $2\Sigma_0 \subsetneq 2\Sigma_1$ is equivalent to confirming the long-standing *Sakoda-Sipser conjecture* that $2\mathsf{D} \subsetneq 2\mathsf{N}$ [11,6].

The hierarchy is defined by analogy to the *polynomial-time hierarchy* of standard complexity theory, whose k-th level consists of the classes $\Sigma_k\mathsf{P}$ and $\Pi_k\mathsf{P}$

of languages decided by polynomial-time *alternating Turing machines* (ATMs) where the number of alternations is bounded as above [13,12]. The question whether this hierarchy is strict is, of course, a well-studied open problem, also hosting on its lowest two levels the famous question whether $\mathsf{P} = \mathsf{NP}$.

An important feature of the polynomial-time hierarchy, highlighting its robustness, is that its classes can be defined in several equivalent ways, which are all quite natural but also quite different from each other conceptually. Indeed, apart from their standard definition in terms of polynomial-time ATMs, these classes can also be defined in terms of:

- *Polynomial-time predicates.* For example, a language is in class $\mathsf{\Sigma_1P} = \mathsf{NP}$ iff it consists of every string which can, together with a suitable 'certificate', satisfy a binary predicate which is decided by a *deterministic Turing machine* (DTM) in time polynomial in the length of the string [12].
- *Polynomial-time oracle Turing machines.* For example, a language is in class $\mathsf{\Sigma_2P} = \mathsf{NP}^{\mathsf{NP}}$ iff it is decided by a polynomial-time *nondeterministic Turing machine* (NTM) which has access to an oracle for a language of NP [10,12].
- *Logical formulas.* For example, a language is in $\mathsf{PH} = \bigcup_{k\geq 0} \mathsf{\Sigma}_k\mathsf{P}$ iff it consists of every string which satisfies a formula in *second-order logic* [2,4].

It is natural to ask whether the classes of the polynomial-size hierarchy also admit analogous alternative definitions, next to their original one in terms of polynomial-size 2AFAs. That is, what kind of (i) 'polynomial-size predicates', (ii) 'polynomial-size oracle two-way finite automata', and (iii) logical formulas match 2AFAs with polynomially many states and finitely many alternations?

In this article we study (i). We identify a proper definition for *polynomial-size predicates* such that suitably quantified predicates of this kind characterize the classes $2\mathsf{\Sigma}_k$ and $2\mathsf{\Pi}_k$, for all k. Starting with the case $k = 1$, we experiment with several natural ways of defining predicates which characterize $2\mathsf{\Sigma}_1 = 2\mathsf{N}$, namely the (families of) languages decided by polynomial-size 2NFAs. After we reach the correct definition for this class, we generalize for all classes of the hierarchy.

This settles part (i). Part (ii) remains open: We know of no model of 'oracle two-way finite automaton' for characterizing the classes of the polynomial-size hierarchy. As for (iii), a partial answer was given in [8], where a class of suitably structured formulas of *monadic second-order logic with successor* were proven equivalent to polynomial-size *sweeping* 2NFAs (i.e., 2NFAs which turn their head only on the endmarkers) when the length is polynomial and certain structural parameters are appropriately bounded; the full answer involves suitably structured formulas of *first-order logic with successor and transitive closure* [9].

1.1 Preparation

If $n \geq 0$, then $[n] := \{0, 1, \ldots, n{-}1\}$. If Σ is an *alphabet* and the symbols $\vdash,\dashv \notin \Sigma$ are *endmarkers*, then $\Sigma_\mathrm{e} := \Sigma \cup \{\vdash,\dashv\}$. If $z \in \Sigma^*$ is a string over Σ, then $|z|$ is its length and z_i is its i-th symbol, if $1 \leq i \leq |z|$; or $\vdash$, if $i = 0$; or $\dashv$, if $i = |z|{+}1$. A language $L \subseteq \Sigma^*$ is *decided* (or *solved*) by a machine M if M accepts exactly the strings in L. A language family[1] $(L_h)_{h\geq 1}$ is *decided* (or *solved*) by a family of machines $(M_h)_{h\geq 1}$ if every M_h solves L_h. A family of automata $(M_h)_{h\geq 1}$ is *polynomial-size* if M_h has $\leq p(h)$ states, for some polynomial p and all h.

A *two-way alternating finite automaton* (2AFA) is a tuple $M = (Q, U, \Sigma, \delta, q_s)$, where Q is a set of *states*, Σ is an *alphabet*, and $\delta \subseteq Q \times \Sigma_e \times Q \times \{\text{L,R}\}$ is the *transition relation*, for L,R two direction-indicating tags; one state q_s is *special* (start/accept) and each state is *universal*, if in $U \subseteq Q$, or *existential*, if in $Q \setminus U$.

An input $z \in \Sigma^*$ is presented on the tape between the endmarkers, as $\vdash z \dashv$. The automaton starts at q_s and on $\vdash$. Whenever at a state p and on a symbol a, it switches to state q and moves its head in direction d, for every q and d such that $(p, a, q, d) \in \delta$ —never violating an endmarker, except to move off $\dashv$ into q_s. The result is a tree of *configurations*, i.e., state-position pairs from $Q \times \{0, \ldots, |z|+2\}$, with $(q_s, 0)$ as root; we call this tree the *computation of M on z*, $\text{COMP}_M(z)$.

The unique *accepting* configuration is $(q_s, |z|+2)$. A *rejecting* configuration is any (p, i) where $i \leq |z|+1$ and δ contains no tuple of the form $(p, z_i, \cdot, \cdot)$. The accepting and rejecting configurations are called *halting*. A non-halting configuration (p, i) is *existential* or *universal*, according to what p is; it is also called *deterministic*, if δ contains exactly 1 tuple of the form $(p, z_i, \cdot, \cdot)$.

A *full computation path* in $\text{COMP}_M(z)$ is any path π which starts at the root and is infinite (*looping*) or ends at a leaf (*halting*); in the latter case, π is either *accepting* or *rejecting*, according to what the leaf is. A *full computation tree* in $\text{COMP}_M(z)$ is any subtree τ such that (1) τ contains the root, (2) each existential configuration in τ has exactly 1 of its children in τ, and (3) each universal configuration in τ has all of its children in τ. We call τ *looping*, if it is infinite; *accepting*, if it is finite and all its leaves are accepting; and *rejecting*, otherwise. If $\text{COMP}_M(z)$ contains an accepting full computation tree, then *M accepts z*.

Let $k \geq 1$. If every full computation path in $\text{COMP}_M(z)$ for any z switches $< k$ times between existential and universal configurations, we say M is a $2\Sigma_k$FA, if $q_s \notin U$, or a $2\Pi_k$FA, if $q_s \in U$ —a $2\Sigma_1$FA is also called *nondeterministic* (a 2NFA). If every non-halting configuration ever exhibited by M is actually deterministic, we say M is a $2\Sigma_0$FA or a $2\Pi_0$FA or simply *deterministic* (a 2DFA). If δ never uses the L tag, we say M is *one-way* (1AFA, 1NFA, 1DFA).

Let $k \geq 0$. The class $2\Sigma_k$ (respectively, $2\Pi_k$) consists of every language family which is solved by a polynomial-size family of $2\Sigma_k$FAs (respectively, $2\Pi_k$FAs):

$$2\Sigma_k := \left\{ (L_h)_{h\geq 1} \;\middle|\; \begin{array}{l} \text{there exists a } 2\Sigma_k\text{FAs family } (M_h)_{h\geq 1} \text{ and a polynomial } p \\ \text{such that every } M_h \text{ solves } L_h \text{ with } \leq p(h) \text{ states.} \end{array} \right\},$$

and similarly for $2\Pi_k$. Easily, $2\Sigma_k, 2\Pi_k \subseteq 2\Sigma_{k+1}, 2\Pi_{k+1}$ for all k. We also write 2D for $2\Sigma_0 = 2\Pi_0$; 2N for $2\Sigma_1$; and 2H for $\cup_{k\geq 0} 2\Sigma_k = \cup_{k\geq 0} 2\Pi_k$.

2 The Case of 2N

The class 2N is the minicomplexity analogue of NP. The predicate characterization of NP is given by the following well-known fact (which uses Def. 1):

[1] For an example of a language family, see $\text{TWL} = (\text{TWL}_h)_{h\geq 1}$ on page 237.

Theorem 1. *A language L is in* NP *iff there exists a polynomial-time binary predicate R such that, for all x: $x \in L \iff (\exists y)R(x,y)$.*[2]

Definition 1. *A binary predicate R is* polynomial-time *if there is a* DTM *M and a polynomial p such that, for all x,y: $R(x,y) \iff M$ accepts $\langle x,y \rangle$ in time $p(|x|)$.*

E.g., if L is SAT (the *satisfiability problem* [12]), then R is the predicate which is true whenever x is a Boolean formula (the *instance*) and y is a truth-assignment which satisfies it (the *certificate*); M is the DTM which computes the value of x under y and accepts iff the result is "true"; and p is the small polynomial which bounds the time spent by M as a function of the length of x.

Our goal is to replicate this setting for 2N. That is, we want a characterization of 2N as captured by the following statement and definition:

Theorem 2. *A language family $(L_h)_{h\geq 1}$ is in* 2N *iff there exists a polynomial-size binary predicate family $(R_h)_{h\geq 1}$ such that, for all h and all x:*

$$x \in L_h \iff (\exists y)R_h(x,y)\,.$$

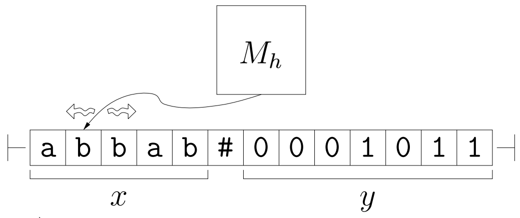
Definition 2. *A binary predicate family $(R_h)_{h\geq 1}$ is* polynomial-size *if there exists a family of 'deterministic finite-state acceptors' $(M_h)_{h\geq 1}$ and a polynomial p such that, for all h and all x,y:*

$$M_h \text{ has } \leq p(h) \text{ states} \qquad \& \qquad R_h(x,y) \iff M_h \text{ accepts } \langle x,y \rangle.$$

E.g., if L_h is TWL$_h$ (the *two-way liveness problem* on h-tall graphs [6,7]), then R_h should be the predicate which is true whenever x is a string of h-tall two-column graphs and y is a path from the leftmost to the rightmost column of the respective multi-column graph; M_h should be some kind of a deterministic finite-state machine which scans the arrows of y and accepts iff they are all present in the graph of x, the first one departs from the leftmost column, and the last one arrives at the rightmost column; and p should be a polynomial bounding the number of states needed to perform these checks.

All we need to do, in order to complete this setting, is to clarify what type of acceptors we should use in Def. 2 so that Th. 2 holds. We explore our options in the next sections. We start with two naive attempts, and explain why they fail. We then continue with a more educated guess which, although it fails, too, it captures a different minicomplexity class. The correct choice is given in Sect. 2.3.

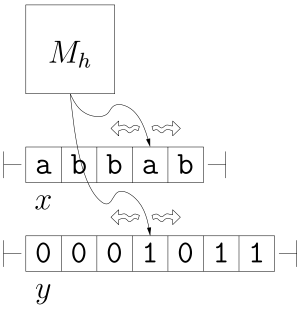
2.1 Two Naive Attempts



The straightforward attempt is to simply have each M_h be a 2DFA which receives the pair $\langle x,y \rangle$ on its input tape as the #-delimited concatenation x#y. But this model is too weak. Intuitively, to check $R_h(x,y)$, M_h must compare corresponding symbols of x and y (i.e., symbols around x_i with symbols around y_i), a task which is impossible for a finite-state machine when x and y become arbitrarily long.

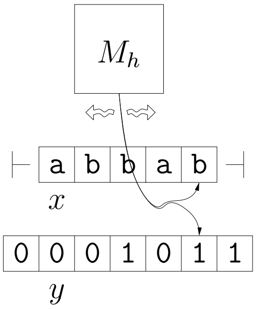
[2] Note that y need never be more than polynomially long, as R is polynomial-time.

To enable M_h to compare corresponding symbols of x and y, we may place x and y on different tapes, each with its own, independent, two-way head. Formally, $M_h = (Q, \Sigma, \Delta, \delta, q_{\mathrm{s}})$, where Σ and Δ are the alphabets for instances and certificates, respectively, and the transition function has the form $\delta : Q \times \Sigma_{\mathrm{e}} \times \Delta_{\mathrm{e}} \longrightarrow Q \times \{\textsc{l},\textsc{r}\} \times \{\textsc{l},\textsc{r}\}$. But now the model is too strong: M_h can use the distance between $\vdash$ and the head on the second tape as counter to solve problems that are even non-regular.



2.2 A Better Attempt

To fix our problems, we must prevent M_h from using its second head as counter. One way to do this, is to first require that x and y are (almost) equally long, then remove the ability of the heads to move independently. Formally, we require that $|y| = |x| + 2$ and $\delta : Q \times \Sigma_{\mathrm{e}} \times \Delta \longrightarrow Q \times \{\textsc{l},\textsc{r}\}$. Let us call this type of machine a *synchronous two-way deterministic finite verifier* (2DFV$_*$). It looks promising.

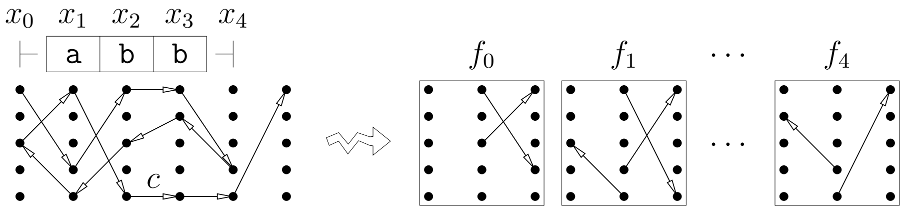


For one, we can now prove the forward direction of Th. 2. It follows from the next lemma, when we apply it to every member of a family $(L_h)_{h \geq 1} \in 2\mathsf{N}$.

Lemma 1. *If L is solved by an s-state* 2NFA, *then some binary predicate R is solved by an s-state* 2DFV$_*$ *and is such that, for all x: $x \in L \iff (\exists y)R(x,y)$.*

Proof. Let $N = (Q, \Sigma, \delta, q_{\mathrm{s}})$ be the 2NFA which solves L.

To motivate R, consider any $x \in L$. Let $n := |x|$. Consider any accepting computation of N on x. Remove all cycles from it, to get the corresponding *minimal* accepting computation —call it c. Because c is minimal, its representation in the configuration graph of N on x (i.e., the graph with all configurations in $Q\times\{0,\ldots,n{+}2\}$ as vertices, and all computation steps allowed by δ as arrows) is a path where no two arrows have a common endpoint. Split this $(n{+}3)$-column representation into $n{+}2$ three-column graphs $f_0, f_1, \ldots, f_{n+1}$, one for each column but the last one, where each f_i represents only the steps performed on x_i.



Since no two arrows have a common endpoint, each f_i is really a partial injection from Q to $Q\times\{\textsc{l},\textsc{r}\}$. Let $\Delta := (Q \rightharpoonup Q\times\{\textsc{l},\textsc{r}\})$ be the alphabet of all such partial injections. Then, we can use $y := f_0 f_1 \ldots f_{n+1} \in \Delta^*$ as a certificate for x.

Indeed, define $R \subseteq \Sigma^* \times \Delta^*$ so that $R(x, y)$ holds iff (1) $|x|+2 = |y|$; (2) the $(|x|+3)$-column graph derived from y (by viewing each y_i as a three-column graph; then identifying the last two columns of each y_i with the first two columns of y_{i+1}; then dropping the first column of the leftmost y_i) contains a path from the top of the leftmost column to the top of the rightmost one; and (3) every arrow (p, q, d) of every y_i is a legal step of N on x_i: $(p, q, d) \in y_i \implies (p, x_i, q, d) \in \delta$. Then the argument of the previous paragraph proves that $x \in L \implies (\exists y)R(x, y)$. Conversely, if $R(x, y)$, then (3) means that the path guaranteed by (2) is an accepting computation of N on x, and thus $x \in L$.

Finally, R is solved by the s-state 2DFV$_*$ $M = (Q, \Sigma, \Delta, \delta', q_s)$ which, on input $\langle x,y \rangle$, interprets y as a $(|x|+3)$-column graph as above and follows the unique path out of q_s of the leftmost column, verifying that all arrows in the graph are consistent with δ and that the path terminates at q_s of the rightmost column. Formally, every $\delta'(p, a, f)$ is either $f(p)$, if $f(p)$ is defined and all arrows in f are consistent with δ; or undefined, otherwise. □

To complete the proof of Th. 2, we would need the converse lemma: *If a binary predicate R is solved by an s-state* 2DFV$_*$*, then $L := \{x \mid (\exists y)R(x, y)\}$ is solved by a* poly(s)*-state* 2NFA. However, in trying to prove this claim, one would find it hard to build the desired 2NFA N for L from the given 2DFV$_*$ for R: the natural approach, where N simply guesses y symbol-by-symbol, fails because, upon returning to an input symbol x_i that has been visited before, N would need to re-guess the corresponding y_i identically as in all previous visits.

As a matter of fact, the backward direction of Th. 2 is false:

Lemma 2. *There exists a polynomial-size binary predicate family $(R_h)_{h\geq 1}$ such that the language family $(L_h)_{h\geq 1}$ where $L_h := \{x \mid (\exists y)R_h(x, y)\}$ is not in* 2N.

Proof. For every h, let $R_h \subseteq \{0\}^* \times [2^h]^*$ be a binary predicate such that $R_h(x, y)$ holds only when $x = 0^{2^h-2}$ and y is the ordered string of all symbols of $[2^h]$:

$$y := \boxed{0}\boxed{1}\boxed{2}\boxed{3}\cdots\boxed{2^h{-}2}\boxed{2^h{-}1}$$

A 2DFV$_*$ M_h can solve R_h by focusing on y and checking that (1) it starts with 0; (2) each of the other symbols is derived from its previous one by adding 1; and (3) the last symbol is $2^h{-}1$. To check (2), M_h goes through every pair of successive symbols, y_i and y_{i+1}, and checks that $y_i{+}1 = y_{i+1}$ by zig-zagging h times between the two positions and comparing the binary representations of y_i and y_{i+1} bit-by-bit. Easily, this requires $O(h)$ states, so $(R_h)_{h\geq 1}$ is polynomial-size.

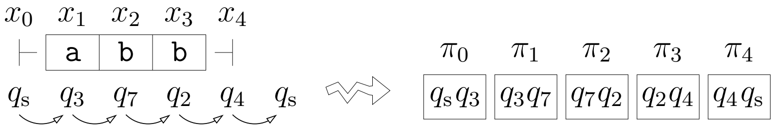
Finally, the only x admitting a certificate under R_h is 0^{2^h-2}, so $L_h = \{0^{2^h-2}\}$, which needs $\geq 2^h{-}2$ states on a 2NFA [1, Fact 5.2]. Hence, $(L_h)_{h\geq 1} \notin$ 2N. □

Overall, our current definitions led us to a strict superset of 2N (Lemmas 1, 2). Before modifying them, let us see which class they really capture. The next two lemmas show that it is the class 2^{1N} corresponding to exponential-size 1NFAs [6].

Lemma 3. *If L is solved by an s-state* 1NFA*, then some binary predicate R is solved by a $O(\log s)$-state* 2DFV$_*$ *and satisfies $x \in L \iff (\exists y)R(x, y)$, for all x.*

Proof. Let $N = (Q, \Sigma, \delta, q_s)$ be the 1NFA which solves L, with $|Q| = s$. Without loss of generality, assume that $Q = [s]$ and that $q_s = 0$. Let $t := \lceil \log_2 s \rceil$.

To motivate R, consider any $x \in L$. Let $n := |x|$. Pick any accepting computation of N on x. This is a list $p_0, p_1, \dots, p_{n+2} \in Q$ such that $p_0 = q_s = p_{n+2}$ and $(p_i, x_i, p_{i+1}, \text{R}) \in \delta$ for all i. Recast this $(n{+}3)$-item list into the list of $n{+}2$ successive pairs $\pi_0, \pi_1, \dots, \pi_{n+1}$, where $\pi_i := (p_i, p_{i+1})$.


x_0 x_1 x_2 x_3 x_4
⊢ a b b ⊣
q_s q_3 q_7 q_2 q_4 q_s
π_0 π_1 π_2 π_3 π_4
q_sq_3 q_3q_7 q_7q_2 q_2q_4 q_4q_s

Now, letting $\Delta := Q \times Q$ be the alphabet of all pairs of states, we can use the string of pairs $y := \pi_0\pi_1 \dots \pi_{n+1} \in \Delta^*$ as a certificate for x.

Therefore, we define $R \subseteq \Sigma^* \times \Delta^*$ so that $R(x, y)$ holds iff (1) $|x|{+}2 = |y|$; (2) y is really a sequence of states (i.e., every two successive symbols are of the form $(\,.\,, p)$ and $(p, \,.\,)$ for some p) from q_s to q_s (the first and last symbols are of the form $(q_s, \,.\,)$ and $(\,.\,, q_s)$, respectively); and (3) this sequence of states is a computation of N on x (i.e., every symbol $y_i = (p, q)$ is a legal step of N on x_i, namely $(p, x_i, q, \text{R}) \in \delta$). Then the argument of the last paragraph shows that $x \in L \Longrightarrow (\exists y)R(x, y)$. Conversely, if $R(x, y)$, then (3) means that the sequence guaranteed by (2) is an accepting computation of N on x, and thus $x \in L$.

Finally, R is solved by a 2DFV$_*$ M which, on input $\langle x,y \rangle$, works as follows. It scans y and, on every two successive symbols $y_i = (p_i, q_i)$ and $y_{i+1} = (p_{i+1}, q_{i+1})$, checks that $q_i = p_{i+1}$ by zig-zagging t times between y_i and y_{i+1} to test that the corresponding bits of $q_i, p_{i+1} \in [s]$ are identical. At the start and end of the scan, M also checks that the first and last symbols of y have respectively the form $(0, \,.\,)$ and $(\,.\,, 0)$. This confirms condition (2). Condition (3) is checked in the same scan: whenever M reads a new symbol $y_i = (p_i, q_i)$, it also verifies that $(p_i, x_i, q_i, \text{R}) \in \delta$. Easily, M needs no more than $O(t) = O(\log s)$ states. □

Lemma 4. *If a binary predicate R is solved by an s-state* 2DFV$_*$*, then the language $L := \{x \mid (\exists y)R(x, y)\}$ is solved by a $2^{O(s)}$-state* 1NFA.

Proof. Let $M = (Q, \Sigma, \Delta, \delta, q_s)$ be the 2DFV$_*$ which solves R, with $|Q| = s$.

Pick any $x \in \Sigma^*$. Let $n := |x|$. To check whether $x \in L$, a 1NFA N guesses a $(n{+}2)$-long $y \in \Delta^*$ and an accepting computation of M on x and the guessed y. The certificate is guessed one symbol per step, as N scans x on its tape; likewise, the accepting computation is guessed one *frontier* per step [5, p. 547].

Formally, $N := (Q', \Sigma, \delta', F_s)$ for $Q' := \{(U, V) \mid U, V \subseteq Q \;\;\&\;\; |U|{+}1 = |V|\}$ the set of all frontiers of M and $F_s := (\emptyset, \{q_s\})$. When at a state (U, V) reading an input symbol $a \in \Sigma_e$, the automaton guesses a corresponding certificate symbol $b \in \Delta$, together with a frontier (U', V') such that (U, V) is (a, b)-*compatible* to it (with respect to δ [5, Def. 2]), and moves to state (U', V'):

$$\big((U,V), a, (U', V'), \text{R}\big) \in \delta' \iff (\exists b \in \Delta)\big[(U,V) \text{ is } (a, b)\text{-compatible to } (U', V')\big]\,.$$

Therefore, N accepts x iff there exists a sequence of guesses $b_i, (U_{i+1}, V_{i+1})$ for $i = 0, 1, \dots, n{+}1$ such that the sequence of frontiers $F_s = (U_0, V_0)$, (U_1, V_1), $\dots$,

$(U_{n+1}, V_{n+1}), (U_{n+2}, V_{n+2}) = F_s$ *fits* the string $(\vdash, b_0)(x_1, b_1)\cdots(x_n, b_n)(\dashv, b_{n+1})$ of symbols over $\Sigma_e \times \Delta$ [5, Def. 3], and thus contains an accepting computation of M on $\langle x, b_0 b_1 \cdots b_{n+1}\rangle$ [5, Lemma 2 and converse]. Hence, N accepts x iff there exists $y \in \Delta^*$ and an accepting computation of M on $\langle x, y\rangle$; i.e., iff $(\exists y)R(x, y)$.

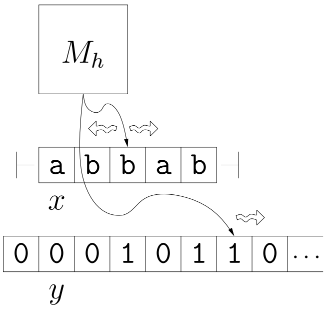
Finally, the number of states of N is $\binom{2s}{s+1} = 2^{O(s)}$ [5, p. 552]. □

Theorem 3. *A language family $(L_h)_{h\geq 1}$ is in 2^{1N} iff there exists a binary predicate family $(R_h)_{h\geq 1}$ which is solved by a polynomial-size family of* 2DFV$_*$s *and is such that, for all h and all x: $x \in L_h \iff (\exists y)R_h(x, y)$.*

By similar arguments, we can also characterize the class 1N corresponding to polynomial-size 1NFAs in terms of *synchronous one-way deterministic finite verifiers* (1DFV$_*$s), the restriction of 2DFV$_*$s where the heads move only forward.

Theorem 4. *A language family $(L_h)_{h\geq 1}$ is in 1N iff there exists a binary predicate family $(R_h)_{h\geq 1}$ which is solved by a polynomial-size family of* 1DFV$_*$s *and is such that, for all h and all x: $x \in L_h \iff (\exists y)R_h(x, y)$.*

2.3 The Right Choice

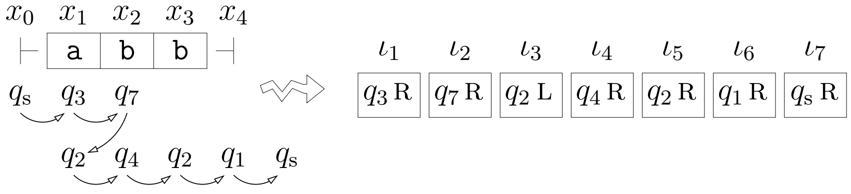


To fix our problems, we must restore M_h's ability to move its heads independently, but still prevent the use of the second head as counter. One way to do this, is to have the second head be one-way. Formally, $\delta : Q \times \Sigma_e \times \Delta \longrightarrow Q \times \{\text{L,R}\}$ again, but now L,R indicate only the first head's motion; the second head moves always right. Let us call this a *two-way deterministic finite verifier* (2DFV).

Now we can finally prove Th. 2. It follows from the next two lemmas.

Lemma 5. *If L is solved by an s-state* 2NFA*, then some binary predicate R is solved by an s-state* 2DFV *and is such that, for all x: $x \in L \iff (\exists y)R(x, y)$.*

Proof. Let $N = (Q, \Sigma, \delta, q_s)$ be the 2NFA which solves L. To motivate R, pick any $x \in L$. Pick any accepting computation c of N on x. Let m be its length. The 'instructions' followed by N along c are the pairs $\iota_1, \ldots, \iota_m \in Q \times \{\text{L,R}\}$, where $\iota_i := (q, d)$ iff in the i-th step N switched to q and moved its head towards d.



x_0 x_1 x_2 x_3 x_4
⊢ a b b ⊣
q_s q_3 q_7
q_2 q_4 q_2 q_1 q_s
ι_1 ι_2 ι_3 ι_4 ι_5 ι_6 ι_7
q_3 R
q_7 R
q_2 L
q_4 R
q_2 R
q_1 R
q_s R

Hence, letting $\Delta := Q \times \{\text{L,R}\}$, we can use $y := \iota_1 \cdots \iota_m \in \Delta^*$ as certificate for x.

So, we define $R \subseteq \Sigma^* \times \Delta^*$ so that $R(x, y)$ holds iff the list of state-position pairs derived from $(q_s, 0)$ by following the instructions $y_1, \cdots, y_m \in \Delta$ is an accepting computation of N on x. It should be clear that $x \in L \iff (\exists y)R(x, y)$.

Moreover, R is solved by the 2DFV $M = (Q, \Sigma, \Delta, \delta', q_s)$ which, on input $\langle x,y \rangle$, simply follows the instructions in y and accepts iff they lead it off $\dashv$ into q_s and never violate δ: when at state p reading $a \in \Sigma_e$ and $(q,d) \in \Delta$, it checks that $(p,a,q,d) \in \delta$ and, if so, switches to q and moves towards d. Easily, M accepts $\langle x,y \rangle$ iff y causes an accepting computation of N on x; i.e, iff $R(x,y)$. $\square$

Lemma 6. *If a binary predicate R is solved by an s-state* 2DFV*, then the language $L := \{x \mid (\exists y)R(x,y)\}$ is solved by an s-state* 2NFA*.*

Proof. Let $M = (Q, \Sigma, \Delta, \delta, q_s)$ be the 2DFV which solves R. Pick any $x \in \Sigma^*$. To check that $x \in L$, a 2NFA $N := (Q, \Sigma, \delta', q_s)$ simulates M on $\langle x,y \rangle$, for $y \in \Delta^*$ a certificate which is guessed on the fly, symbol-by-symbol. When at state p reading symbol $a \in \Sigma_e$, the automaton guesses the next symbol $b \in \Delta$ on the certificate tape, then switches to q and moves towards d, where $(q,d) = \delta(p,a,b)$. Formally, $(p,a,q,d) \in \delta' \iff (\exists b \in \Delta)[(q,d) = \delta(p,a,b)]$.

Easily, N accepts x iff there is a sequence of guesses $b_0, b_1, \dots, b_m$ such that M accepts $\langle x, b_0 b_1 \cdots b_m \rangle$; namely, iff there exists $y \in \Delta^*$ such that $R(x,y)$. $\square$

Note that all our certificates are *finite* strings, which makes sense for $2\Sigma_1$. But we may also work with *infinite* certificates: easily, Lemmas 5 and 6 (and Th. 2) hold even when $R \subseteq \Sigma^* \times \Delta^\omega$, where $\Delta^\omega :=$ {all infinite strings over Δ}, and 2DFVs have infinite certificate tape. This variation of our definitions is optional for $2\Sigma_1$; however, for $2\Pi_1$ and for general $2\Sigma_k, 2\Pi_k$ it is essential.

3 The General Case

We now turn to classes $2\Sigma_k$ and $2\Pi_k$ for arbitrary k. For concreteness, we treat only $2\Sigma_3$. (Our proof does generalize to $2\Sigma_k$, it is straightforward but tedious; then, $2\Pi_k$ is handled by a dual argument.) So, our goal is to prove the following.

Theorem 5. *A language family $(L_h)_{h\geq 1}$ is in $2\Sigma_3$ iff there is a polynomial-size quaternary predicate family $(R_h)_{h\geq 1}$ such that, for all h and all x:*

$$x \in L_h \iff (\exists z_1)(\forall z_2)(\exists z_3)R_h(x, z_1, z_2, z_3)\,.$$

Definition 3. *A quaternary predicate family $(R_h)_{h\geq 1}$ is* polynomial-size *if some family of* 2DFVs *$(M_h)_{h\geq 1}$ and polynomial p are such that, for all h and x,z_1,z_2,z_3:*

$$M_h \textit{ has } \leq p(h) \textit{ states} \qquad \& \qquad R_h(x, z_1, z_2, z_3) \iff M_h \textit{ accepts } \langle x, z_1, z_2, z_3 \rangle.$$

Now, each predicate relates a *finite* string x with three *infinite* strings z_1, z_2, z_3. Accordingly, a 2DFV M has three *infinite* certificate tapes, one per z_j, with its own head h_j. Crucially, the heads are *used in order*: first, M reads from h_1, keeping h_2, h_3 stationary; later, it deactivates h_1 and starts reading from h_2, keeping h_3 stationary; eventually, it deactivates h_2 too, and starts reading from h_3. Formally, $M = (Q, J, \Sigma, \Delta, \delta, q_s)$, where again $\delta : Q \times \Sigma_e \times \Delta \longrightarrow Q \times \{\text{L,R}\}$ but now the single certificate symbol always comes from the currently active head; and $J \subseteq Q$ is the states which cause a jump to the next certificate tape: entering $q \in J$ causes M to deactivate the currently active head h_j and activate h_{j+1}.

As usual, the proof consists of two lemmas, each for a single direction and h.

Lemma 7. *If L is solved by an s-state* 2Σ$_3$FA*, then some quaternary predicate R is solved by an $O(s)$-state* 2DFV *and is such that, for all x:*

$$x \in L \iff (\exists z_1)(\forall z_2)(\exists z_3)R(x, z_1, z_2, z_3)\,.$$

Proof idea. Let $A = (Q, ., \Sigma, ., .)$ be a $2\Sigma_3$FA for L. Then A follows 'instructions' from $\Delta := Q \times \{\text{L,R}\}$ and we define $R \subseteq \Sigma^* \times (\Delta^\omega)^3$ so that $R(x, z_1, z_2, z_3)$ iff *either* $z_1 z_2 z_3$ is the list of instructions followed along an accepting full computation path on x, and z_2 consists of those followed from universal configurations; *or* $z_1 z_2 z_3$ starts as such a list, but contains an invalid instruction in z_2. □

Lemma 8. *If a quaternary predicate R is solved by an s-state* 2DFV*, then the language $L := \{x \mid (\exists z_1)(\forall z_2)(\exists z_3)R(x, z_1, z_2, z_3)\}$ is solved by a $3s$-state* $2\Sigma_3$FA.

Proof idea. Let $M = (Q, ., \Sigma, \Delta, ., q_\text{s})$ be a 2DFV for R. Then a $2\Sigma_3$FA $A := (Q_1 \cup Q_2 \cup Q_3, Q_2, \Sigma, ., q_\text{s}^1)$, where each $Q_j := \{p^j \mid p \in Q\}$ is a copy of Q, checks whether an input $x \in \Sigma^*$ is in L by simulating M on $\langle x, z_1, z_2, z_3\rangle$, for some strings $z_1, z_2, z_3 \in \Delta^\omega$ which are respectively guessed, universally selected, and guessed, each of them up to some prefix and on the fly. This works in three phases, where each phase j uses states exclusively from Q_j. □

References

1. Birget, J.-C.: Two-way automata and length-preserving homomorphisms. Mathematical Systems Theory 29, 191–226 (1996)
2. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In: Karp, R.M. (ed.) Complexity of Computation. AMS-SIAM Symposia in Applied Mathematics, vol. VII, pp. 43–73 (1974)
3. Geffert, V.: An alternating hierarchy for finite automata. Theoretical Computer Science 445, 1–24 (2012)
4. Immerman, N.: Descriptive complexity. Springer (1998)
5. Kapoutsis, C.: Removing bidirectionality from nondeterministic finite automata. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 544–555. Springer, Heidelberg (2005)
6. Kapoutsis, C.: Size complexity of two-way finite automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 47–66. Springer, Heidelberg (2009)
7. Kapoutsis, C.A.: Minicomplexity. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 20–42. Springer, Heidelberg (2012)
8. Kapoutsis, C., Lefebvre, N.: Analogs of Fagin's Theorem for small nondeterministic finite automata. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 202–213. Springer, Heidelberg (2012)

9. Kapoutsis, C., Mulaffer, L.: A descriptive characterization of the power of small 2NFAs (in preparation, 2014)
10. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proceedings of the Symposium on Switching and Automata Theory, pp. 125–129 (1972)
11. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proceedings of STOC, pp. 275–286 (1978)
12. Sipser, M.: Introduction to the theory of computation, 3rd edn. Cengage Learning (2012)
13. Stockmeyer, L.J.: The polynomial-time hierarchy. Theoretical Computer Science 3, 1–22 (1976)