Math. Systems Theory 26, 237–269 (1993)

Mathematical Systems Theory

State-Complexity of Finite-State Devices, State Compressibility and Incompressibility

Jean-Camille Birget

Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE 68588-0115, USA
birget@cse.unl.edu

Abstract. We study how the number of states may change when we convert between different finite-state devices. The devices that we consider are finite automata that are one-way or two-way, deterministic or nondeterministic or alternating. We obtain several new simulation results (e.g., an n-state 2NFA can be simulated by a 1NFA with $\leq 8^n + 2$ states, and by a 1AFA with $\leq n^2$ states), and state-incompressibility results (e.g., in order to simulate an n-state 2DFA, a 1NFA needs $\geq \sqrt{2^{n-2}}$ states, and a 2AFA needs $\geq c\sqrt{n}$ states for some constant c, in general).

1. Definitions and Notation

We consider several kinds of finite-state devices; all of them can recognize only regular languages, but some may need fewer states than others to recognize the same language. The most standard finite-state devices are the (*one-way*) *deterministic finite automata* (abbreviated "**1DFA**," see the Appendix, [RS], and pp. 13–19 of [HU]), and the (*one-way*) *nondeterministic finite automata* (abbreviated "**1NFA**," see [RS] and pp. 19–24 of [HU]). Here we use *partial* 1DFAs (i.e., the next state could be undefined) and *complete* 1DFAs (there is always exactly one next state); we always indicate which kind of 1DFA we are using. We also consider two-way finite automata, deterministic (abbreviated "**2DFA**") and nondeterministic (abbreviated "**2NFA**"); they can move their reading head to the left and the right. Early references are [RS] and [Sh], where it is proved that two-way finite automata recognize only regular languages (see also pp. 36–41 of [HU]). Later, *alternating* finite automata were introduced. See pp. 127ff of [CKS] and [BL] for *one-way* alternating finite automata (abbreviated "**1AFA**"); see [LLS] for *two-way*

finite automata (abbreviated "**2AFA**"); our 2AFAs are two-way generalizations of the 1AFAs of [CKS] and [BL]: we allow arbitrary boolean functions (instead of just the AND and OR of [LLS]). We also consider ∀-1NFAs; these are like 1NFAs, but they use a different accepting rule: an input is accepted iff *all* computation paths on that input end in accept states. Precise definitions of the various kinds of finite-state devices used in this paper are given in the Appendix.

Since we are interested in the number of states that a device needs to recognize a language, we consider the following *classes of languages*:

For a positive integer n, and

$$\text{DEVICE} \in \{\text{complete 1DFA, partial 1DFA, 1NFA, 2DFA, 2NFA, 1AFA, ...}\},$$

we let [***n* state DEVICE**] be the set of languages (over all possible finite alphabets) that are recognized by some automaton of type DEVICE with n states. For example, $[n \text{ state 1NFA}] = \{L/\Sigma \text{ is a finite alphabet, } L \subseteq \Sigma^* \text{ and } L \text{ is recognized by a 1NFA with } n \text{ states}\}$. It should be emphasized that we do *not* fix a particular alphabet. We assume however that all our alphabets are finite subsets of some fixed countable set; this never constrains us and it avoids possible set-theoretic difficulties.

Let us define some *operators* that can be applied to any class C of languages:

The *complementation* operator **co-**$(\cdot)$ is defined by

$$\textbf{co-}C = \{\Sigma^* - L/\text{the alphabet of } L \text{ is } \Sigma, \text{ and } L \in C\}.$$

The *reversal* operator $(\cdot)^{\text{rev}}$ is defined by

$$C^{\text{rev}} = \{L^{\text{rev}}/L \in C\} \qquad \text{(see pp. 71–72 of [HU]).}$$

The *closure under length-preserving homomorphisms* **L.P.HOM**$(\cdot)$ is defined by

$$\textbf{L.P.HOM}(C) = \{\varphi(L)/L \in C \text{ and } \varphi \text{ is a length-preserving homomorphism defined on the alphabet of } L\}.$$

A length-preserving homomorphism (abbreviated l.p.hom.) is a function $\varphi: \Sigma^* \to \Delta^*$, where Σ and Δ are finite alphabets, such that:

(1) for all $u, v \in \Sigma^*$: $\varphi(uv) = \varphi(u)\varphi(v)$, and
(2) for all $u \in \Sigma^*$: $|\varphi(u)| = |u|$.

Here $|u|$ denotes the length of the word u. See pp. 60–62 of [HU] for some applications of homomorphisms. Note that a length-preserving homomorphism $\varphi: \Sigma^* \to \Delta^*$ is determined by its restriction $\Sigma \to \Delta$ (a letter-to-letter map).

In this paper we study how the number of states changes when we convert one device into another. A result of the form "the 1AFAs with $\leq g(n)$ states can simulate all n-state 2DFAs" is an *inclusion* result (or a "*compressibility*" result, when $g(n) < n$); we can also write $[n\text{-state 2DFA}] \subseteq [g(n)\text{-state 1AFA}]$ (see, e.g., Corollary 2.4). A result of the form "a 2AFA needs $\geq c\sqrt{n}$ states (for some constant $c > 0$) to simulate an n-state 2DFA (in the worst case)," is an *incompressibility* result; we can also express that fact by writing $[n\text{-state 2DFA}] \subseteq [f(n)\text{-state 2AFA}] \Rightarrow f(n) \geq c\sqrt{n}$ (see Theorem 5.2). An early reference on that

subject is [MF]. In the present paper the number of states is used as the only measure of complexity; this is motivated by the connection between the number of states and the memory (or space) complexity. In [MF] other measures are used as well (e.g., the number of transitions in a 1NFA).

2. Some Earlier Inclusion and Compressibility Results, and Some Consequences

Theorem 2.1 (E. Leiss, J. Brzozowski, D. Kozen). *For all* $n > 0$:

$[n$ state **1AFA**$] = [2^n$ state **complete 1DFA**$]^{rev}$.

The left-to-right inclusion was observed by Kozen [Ko], [CKS]. The other inclusion was proved by Leiss (see [L1], where the theorem is stated), using a construction of [BL]. In [BL] it is shown that n-state 1AFAs (reversed) are equivalent to sequential circuits with n boolean state-variables; this also demonstrates the practical importance of 1AFAs. Compare Theorem 2.1 with the incompressibility results: Fact 3.2, Theorem 3.7, and Fact 3.8.

We have the following immediate consequences:

Corollary 2.2.

(1) $[n$ state complete **1DFA**$] \subseteq [\lceil \log_2 n \rceil + 2$ state **halting & sweeping 2AFA**$]$.
(2) $[n$ state **1AFA**$] \subseteq [2^n + 1$ state **1NFA**$] \cap [2^n + 1$ state $\forall$**1NFA**$]$
$\cap [2^n + 2$ state **halting-&-sweeping 2DFA**$]$.

In the above corollary we mention *halting* and *sweeping* two-way automata. The following definitions are used (in all cases: deterministic, nondeterministic, or alternating).

Definition. A two-way finite automaton is **halting** iff it has no infinite computation on any input. A two-way finite automaton is **sweeping** iff its head makes turnabouts only at the endmarkers of the input tape, for all inputs and all computations (see [Si1]).

Proof of Corollary 2.2. (1) By Theorem 2.1, $[n$ state complete 1DFA$] \subseteq [\lceil \log_2 n \rceil$ state 1AFA$]$. Moreover, if a language L^{rev} is recognized by a 1AFA $\mathbf{A}_1$ with k $(=\lceil \log_2 n \rceil)$ states, then its reverse, namely L, is recognized by the following sweeping and halting 2AFA $\mathbf{A}_2$ (with left endmarker ‹ and right endmarker ›): on an input word w, $\mathbf{A}_2$ first reads ‹w› from left to right using a new state i; when the right endmarker is found $\mathbf{A}_2$ goes to the start states of $\mathbf{A}_1$, reads ›, moves to the left and simulates $\mathbf{A}_1$ while reading ‹w from right to left (i.e., backward); when the left endmarker is found and $\mathbf{A}_2$ is in an accept state of $\mathbf{A}_1$ then $\mathbf{A}_2$ reads ‹ (still from right to left) and goes to a new accept state f. In state f, $\mathbf{A}_2$ reads all of w› from left to right. Now, $\mathbf{A}_2$ accepts w iff $\mathbf{A}_1$ accepts w^{rev}.

(2) By Theorem 2.1, [n state 1AFA] $\subseteq$ [2^n state complete 1DFA]^{rev}. Moreover, if a language L^{rev} is recognized by a 1DFA **A** with k ($=2^n$) states, then its reverse, namely L, is recognized by a 1NFA with $k+1$ states (obtained from the state graph of the 1DFA by reversing all the edges, choosing the start state of **A** as the accept state of the 1NFA, and adding a new start state; this is a classical construction). So, [n state 1AFA] $\subseteq$ [2^n+1 state 1NFA].

Since **A** is a complete DFA, the language $\Sigma^* - L^{rev}$ is also recognized by a complete 1DFA **B** with k states (see p. 59 of [HU] for the classical construction). Since $\Sigma^* - L^{rev} = (\Sigma^* - L)^{rev}$, we can construct a 1NFA for $\Sigma^* - L$ with $k+1$ states, as above. Finally, by complementing this 1NFA we obtain a $\forall$1NFA. So, [n state 1AFA] $\subseteq$ [2^n+1 state $\forall$1NFA].

The proof that [n state 1AFA] $\subseteq$ [2^n+2 state halting-&-sweeping 2DFA] is very similar to the proof in (1). □

Theorem 2.1 provides a new and simple proof of the fact (first proved by Kozen [Ko], [CKS]) that there exist languages which have a 1AFA with n states, but whose minimum complete 1DFA has 2^{2^n} states. Indeed, it is known that there are languages L_m (over an alphabet of size 2) such that L_m^{rev} is recognized by a 1DFA with m states but such that the minimum complete 1DFA for L_m has 2^m states (see Propositions 1 and 2 of [L1]). Then, for $m = 2^n$, we get a language $L_m \in$ [$m = 2^n$ state complete 1DFA]^{rev} = [n state 1AFA]; but L_m^{rev} needs a complete 1DFA with $2^m = 2^{2^n}$ states.

Theorem 2.3.

(1) [Sh] (improved)	[n state **2DFA**] $\subseteq$ [n^n state **partial 1DFA**].
	[n state **2NFA**] $\subseteq$ [$2^{(n-1)^2+n}$ state **complete 1DFA**].
(2) [HU]	[n state **2NFA**] $\subseteq$ [$n!\,(1+\varepsilon(n))$ state **1NFA**], with $\lim \varepsilon(n) = 0$.
(3) [V]	[n state **2NFA**] $\subseteq$ [4^n state $\forall$-**1NFA**].
(4) [LLS] (generalized)	[n state **2AFA**] $\subseteq$ [$2^{(n+1)2^n}$ state **complete 1DFA**].

Proof. (1) Shepherdson [Sh] proves that a 2DFA with n states is equivalent to a partial 1DFA with $n(n+1)^n$ states. His proof easily generalizes to nondeterminism, using relations instead of functions: a 2NFA with n states is equivalent to a partial 1DFA with $(2^n-1)2^{n^2}$ states. See the Appendix (Theorem A3.4) for the improved version.

For (2) and (3) see respectively [HU] and [V].

(4) In [LLS] the result [n state 2AFA] $\subseteq$ [2^{n2^n} state complete 1DFA] is proved for a more special kind of 2AFA (where every state is either deterministic or $\exists$ or $\forall$, instead of an arbitrary boolean function of the states; and the reading head moves only when it is in a deterministic state). See the Appendix for the proof in the general case. □

In Section 4 we improve on (2), replacing $n!\,(1+\varepsilon(n))$ by 8^n+2. Also, compare (2) and (3) with Theorem 5.1 (incompressibility result). We have the following (new) result:

Corollary 2.4. *For all $n \geq 1$ we have:*

(1) $[n$ state **2DFA**$] \subseteq [(n+2)\log_2(n+2)+1$ state **1AFA**$]$.
(2) $[n$ state **2NFA**$] \subseteq [n^2$ state **1AFA**$]$.
(3) $[n$ state **2AFA**$] \subseteq [(n+3)2^{n+2}$ state **1AFA**$]$.

Comments. These results are close to optimal; compare (2) with Theorem 3.7, and (3) with Corollary 3.9, which give incompressibility results.

Proof of Corollary 2.4. (1) By Fact A3.1 in the Appendix we have $[(n+2)^{n+2}$ state partial 1DFA$]^{\text{rev}} \subseteq [(n+2)^{n+2}+1$ state complete 1DFA$]^{\text{rev}}$. By Theorem 2.1, this is contained in $[(n+2)\log_2(n+2)+1$ state 1AFA$]$.

(2) By Fact A3.1 in the Appendix, $[n$ state 2NFA$] \subseteq [n$ state 2NFA$]^{\text{rev}}$. By Theorem 2.3(1), this is contained in $[2^{(n-1)^2+n}$ state complete 1DFA$]^{\text{rev}} \subseteq [2^{n^2}$ state complete 1DFA$]^{\text{rev}} \subseteq_{(\text{by } 2.1)} [n^2$ state 1AFA$]$.

(3) By a proof similar to the one of Corollary 2.2(1) we have

$$[n \text{ state 2AFA}] \subseteq [n+2 \text{ state 2AFA}]^{\text{rev}}.$$

By Theorem 2.3(4) the latter class is contained in

$$[2^{(n+3)2^{n+2}} \text{ state complete 1DFA}]^{\text{rev}} \subseteq_{(\text{by } 2.1)} [(n+3)2^{n+2} \text{ state 1AFA}]. \quad \square$$

Fact 2.5 [B3]. *For all $n > 0$,*

$$\text{L.P.HOM}[n \text{ state } \mathbf{2NFA}] = \text{L.P.HOM}[n \text{ state } \mathbf{2DFA}].$$

Theorem 2.6 [B3]. *There are constants $c \geq 1$ such that, for all n,*

$[n$ state **2AFA**$] \subseteq$ L.P.HOM$[cn^2$ state **2DFA**$]$,
$[n$ state **halting 2AFA**$] \subseteq$ L.P.HOM$[cn$ state **2DFA**$]$,
$[n$ state **1-pebble 2NFA**$] \subseteq$ L.P.HOM$[cn^2$ state **2DFA**$]$.

A *1-pebble 2NFA* is a 2NFA with one marker (pebble) that can be picked up, put down, and left in place, picked up again, etc., by the reading head. See [BH], and also p. 73, ex. 3.19, of [HU], and [B3]. Compare Theorem 2.6 with the incompressibility results in Corollary 5.3.

The closure under length-preserving homomorphisms has many interesting properties; we see some in the next sections. In [B3] it was observed:

(1) NTIME $O(n) \subseteq$ L.P.HOM(AC^1) (this follows from results of Book and Greibach [BG]).
(2) Some *NP-complete* problems are contained in L.P.HOM(NC^1).

(Here AC^1 and NC^1 are the well-known parallel-complexity classes.) From this (and from Theorem 2.6 and Corollary 5.3) it can be seen that the closure under l.p.hom can be much more powerful than nondeterminism.

The class L.P.HOM$[n$ state 2DFA$]$ can be characterized exactly by a machine model (namely, nondeterministic single-tape Turing machines with n states, that visit the same state at most once at the same position, and do not leave the input portion of the tape). See [B3] and [H].

3. Earlier Incompressibility and Nonclosure Results

Fact 3.1. *For all $n > 0$ we have*

If [n state partial **1DFA**] $\subseteq$ [f(n) state **2NFA**], *then* f(n) $\geq n$.

An example of a language whose minimum 1DFA has n states, and for which every 2NFA must also have $\geq n$ states, is $L_n = \{a^{n!-1}\}$ (a one-word language over a one-letter alphabet). For a proof, see [B3].

A fortiori, by Fact 3.1, the conversions 1DFA $\to$ 2DFA, 1DFA $\to$ 1NFA, 2DFA $\to$ 2NFA, 1NFA $\to$ 2NFA, do not allow any state compression in general.

The language $L_n = \{a^{n!-1}\}$ (where $n > 0$) was used in [B3] to disprove a conjecture of Meyer and Fischer [MF, p. 190]; they conjectured that a 1-pebble 2DFA needs $\geq cn$ states (for some constant $c > 0$) to recognize $\{a^{n!-1}\}$; it turns out that $(\ln n)^2 \ln\ln n(1 + \varepsilon(n))$ states suffice, where $\lim \varepsilon(n) = 0$ ("ln" is the natural logarithm). The language $\{a^{n!-1}\}$ also serves to disprove a conjecture of Chrobak [C]; he conjectured that, for unary languages, 1AFAs and 2DFAs are polynomially equivalent, regarding their state complexity; but a 2DFA needs n states to recognize $\{a^{n!-1}\}$, while a 1AFA needs only $\lceil \log_2 n \rceil$ states (by Theorem 2.1); a Π_2-1AFA (in which, by definition, every computation first uses $\forall$-states, then $\exists$-states, then halts) can recognize $\{a^{n!-1}\}$ with $(\ln n)^2 \ln\ln n\,(1 + \varepsilon(n))$ states (where $\lim \varepsilon(n) = 0$); see [B3].

Theorem 2.1 applied to the language $\{a^{2^n-1}\}$ immediately yields:

Fact 3.2. *For all $n > 0$ we have*

If [n state **1AFA**] $\subseteq$ [f(n) state **2NFA**], *then* f(n) $\geq 2^n$.

Compare this with Corollary 2.4.

Proof. The language $\{a^{2^n-1}\}$ has a 1DFA with 2^n states, thus (by Theorem 2.1) a 1AFA with n states. On the other hand, a 2NFA needs $\geq 2^n$ states (see [B3]). □

Theorem 3.3

(1) [SS] *For all $n > 0$ we have*

If [n state **2DFA**] $\subseteq$ [f(n) state **1NFA**] $\cup$ [f(n) state $\forall$**-1NFA**], *then* f(n) $\geq c^{\sqrt{n}}$ *for some constant* $c > 1$.

(2) [C] *For all n large enough we have*

If [n state unary **2DFA**] $\subseteq$ [f(n) state unary **1NFA**], *then* f(n) $\geq e^{\sqrt{n \ln n}}$.

("Unary" means that the alphabet has just one letter.)

In Theorem 5.1 we improve Theorem 3.3(1) to f(n) $\geq \sqrt{2^{n-2}}$. Compare also with Theorems 4.5 and 2.3(3).

Theorem 3.4 ([SS]; see [B6] for a new proof). *For all $n > 0$ we have*

If [n state $\forall$-**1NFA**] $\subseteq$ [f(n) state **1NFA**], *then* f(n) $\geq 2^n$.

A fortiori the conversion of an n-state 1AFA or 2AFA into a 1NFA requires $\geq 2^n$ states.

Theorem 3.5 [Si1]. *For all $n > 0$ we have*

If [n-state **1NFA**] $\subseteq$ [f(n) state **sweeping 2DFA**], *then* f(n) $\geq 2^n$.

Sakoda and Sipser [SS] conjecture that if [n state 1NFA] $\subseteq$ [f(n) state 2DFA] (not necessarily sweeping), then f(n) $>$ p(n) for any polynomial p($\cdot$). This is still an unsolved (and apparently very difficult) problem.

Theorem 3.6 [CS], [Ka]. *For all $n > 0$*

If [n state Σ_2-**1AFA**] $\subseteq$ [f(n) state **2NFA**], *then* f(n) $\geq c^n$ *for some constant* $c > 1$.

A Σ_2-1AFA is a 1AFA which only has $\forall$- and $\exists$-states, and in which every computation first encounters $\exists$-states, then $\forall$-states, and then halts; see the Appendix for more details. Theorem 3.6 also holds if "Σ_2-1AFA" is replaced by "**1-pebble 2DFA**" (this follows from Proposition 3 of [MF]).

Theorem 3.7 [L2]. *For all $n > 0$ we have*

If [n state **1DFA**] $\subseteq$ [f(n) state **1AFA**], *then* f(n) $\geq n$.

In words: in general a 1DFA cannot be compressed by converting to a 1AFA. This is in contrast to Theorem 2.1, which says that the *reverse* of a 1DFA can always be logarithmically compressed by converting to a 1AFA.

The examples of [L1] (referred to after Corollary 2.2) give the following classical incompressibility result (see [MF]), with slightly improved bounds:

Fact 3.8. *For all $n > 0$ we have*

If [n state **1DFA**]$^{\text{rev}}$ $\subseteq$ [f(n) state **1DFA**], *then* f(n) $\geq 2^n$.

By Theorem 3.7, 2DFAs, 1NFAs, and 2NFAs cannot be compressed either, as one converts to 1AFAs. An interesting consequence of Theorem 3.7 is the following new result:

Corollary 3.9. *For all $n > 0$ we have*

If [n-state **2AFA**] $\subseteq$ [g(n) state **1AFA**], *then* g(n) $\geq 2^{n-2}$.

(*This also holds when "2AFA" is replaced by* **sweeping-and-halting 2AFA**.)

Proof. By Corollary 2.2, [2^{n-2} state 1DFA] $\subseteq$ [n state 2AFA] ($\subseteq$[g(n) state 1AFA]). Then, applying Theorem 3.7 to [2^{n-2} state 1DFA] $\subseteq$ [g(n) state 1AFA] we immediately get $g(n) \geq 2^{n-2}$. □

Note. Corollary 3.9 is close to optimal; compare with Corollary 2.4.

4. New Inclusion and Compressibility Results

First we give some results involving length-preserving homomorphisms. Then we use those results to improve the known results about the simulation of 2NFAs and 2AFAs by 1NFAs.

Theorem 4.1. *For every $n > 0$ we have*:

(1) [n-state **1NFA**] $\subseteq$ L.P.HOM[$2\lceil \log_2(n+1) \rceil + 2$ state $\forall$-**1NFA**].
(2) [n-state **1NFA**] $\subseteq$ L.P.HOM[$2\lceil \log_2 n \rceil + 1$ state **halting-&-sweeping 2DFA**].

The $\forall$-1NFAs *in* (1) *are very special*: *only their start state is nondeterministic* (*of* $\forall$-*type*); *all other states are deterministic* (so they are similar to the parallel intersection machines of [SS]).

Corollary.
[n state **1AFA**] $\subseteq$ L.P.HOM[**$2n + 4$** state $\forall$-**1NFA**]
$\cap$ L.P.HOM[**$2n + 3$** state **halting-&-sweeping 2DFA**].

Proof of the Corollary. By Corollary 2.2, [n state 1AFA] $\subseteq$ [$2^n + 1$ state 1NFA]; by Theorem 4.1 this is contained in L.P.HOM[$2\lceil \log_2(2^n + 1 + 1) \rceil + 2$ state $\forall$-1NFA] and in L.P.HOM[$2\lceil \log_2(2^n + 1) \rceil + 1$ state halting-&-sweeping 2DFA]. The corollary then follows. □

Proof of Theorem 4.1. Let L ($\subseteq \Sigma^*$) be recognized by a 1NFA $N = (Q, \Sigma, \delta, q_0, F)$ with $|Q| = n$. We first use a classical construction to obtain an alphabet Δ, a language $L_0 \subseteq \Delta^*$, and a length-preserving homomorphism $h: \Delta^* \to \Sigma^*$ such that $L = h(L_0)$. This construction also works if more general 1NFAs that have a *set* of start states, rather than a single start state, are used.

We take $\Delta = \{(p, a, q) \in Q \times \Sigma \times Q / q \in \delta(p, a)\}$; thus $|\Delta| = |\delta| \leq |\Sigma| \cdot n^2$ (so the alphabet size grows only polynomially in n). We define h by $h(p, a, q) = a$. We take L_0 to be $\{(p_1, a_1, q_1)(p_2, a_2, q_2) \cdots (p_k, a_k, q_k) \in \Delta^* / k \geq 1,\ p_1 = q_0$ (the start state of N), $q_k \in F$, and $q_i = p_{i+1}$ for $i = 1, \ldots, k-1\}$ ($\cup\{\varepsilon\}$ if $q_0 \in F$).

Clearly, then, $L = h(L_0)$. The language L_0 is a *local language*:

$$L_0 = A\Delta^* \cap \Delta^* B - \Delta^* C \Delta^*$$

for some sets $A, B \subseteq \Delta$ and $C \subseteq \Delta^2$ (namely, in this case, $A = \Delta \cap \{q_0\} \times \Sigma \times Q$, $B = \Delta \cap Q \times \Sigma \times F$, and $C = \{(p, a, q)(p', b, q') \in \Delta^2 / q \neq p'\}$). See pp. 26–29 of [E]. Note that L_0 is recognized by the following partial 1DFA with n states: the set of states is Q, the start state is q_0, the set of accept states is F (so far everything

is like in the original 1NFA N); the alphabet is Δ; the next-state (partial) function δ' is defined by $\delta'(r, (p, a, q)) = q$ if $r = p$ (undefined if $r \neq p$).

Theorem 4.1 is proved by showing that L_0 can be recognized:

(1) by a $\forall$-1NFA with $2\lceil \log_2(n+1) \rceil + 2$ states, and
(2) by a halting-&-sweeping 2DFA with $2\lceil \log_2 n \rceil + 1$ states.

We first prove (2).

Proof of (2). We may think of the set Q as a subset of $\{0,1\}^k$ where $k = \lceil \log_2 n \rceil$ (e.g., we view Q as $\{0, 1, \ldots, n-1\}$, represented as binary strings of length $\lceil \log_2 n \rceil$). Then $q = p$ iff at every position the binary digits in the representation of p, respectively q, are the same. Actually the alphabet Δ ($\subseteq Q \times \Sigma \times Q$) really remains the same as before—although now we *think* of the elements of Q as binary strings.

Let $w \in \Delta^*$ be an input string. The 2DFA sweeps over this string $\lceil \log_2 n \rceil$ times. Let us assume $n > 2$. In sweep number i (with $0 \leq i < \lceil \log_2 n \rceil$) the 2DFA checks that, for *all* pairs of adjacent letters $(p, a, q)(p', b, q')$ in w, we have the ith bit of q = the ith bit of p'. Sweep number i goes from left to right if i is even, and from right to left if i is odd (recall that we start with $i = 0$). In sweep number i only two states are used: $(i, 0)$ and $(i, 1)$. If i is even (left-to-right sweep) the state remembers whether in the last letter (p, a, q) read, the ith bit of q is 0 or 1; if i is odd (right-to-left sweep) the state remembers whether in the last letter (p', b, q') read, the ith bit of p' is 0 or 1. This leads to $2\lceil \log_2 n \rceil$ states. If $\lceil \log_2 n \rceil$ is even an additional sweep is needed, and an additional (accept) state is introduced to make the reading head end up at the right end of the tape, in case the input is accepted.

We need some special care at the ends of the input.

At the left end, we begin in the start state (0, 0), read the left endmarker and stay in state (0, 0). If the next letter is (q_0, a, q) (where q_0 is the start state of N), we go to state (0, 0th bit of q); the sweep then continues as stated before. If $p \neq q_0$ the 2DFA halts (and rejects). Similarly, when i is even, the ith sweep starts at the left end in state $(i, 0)$, reads the left endmarker and stays in state $(i, 0)$; then (q_0, a, q) is read, and the next state is (i, ith bit of q); the sweep then continues. When i is odd and the left endmarker is reached (from the right) in state $(i, ?)$, we read the endmarker and go to state $(i + 1, 0)$.

At the right end: when the right endmarker is reached in state $(i, ?)$ (with i even), the 2DFA reads the endmarker and goes to state $(i + 1, 1)$ if $1 < i + 1 < \lceil \log_2 n \rceil$, or halts if $i + 1 = \lceil \log_2 n \rceil$. At the right end of the tape, at the beginning of the sweep number i (when i is odd), the 2DFA is in state $(i, 1)$, reads the right endmarker, and stays in state $(i, 1)$. If the next letter is of the form (p, a, q) with $q \in F$, the 2DFA goes to state (i, ith bit of p) after reading this letter; if the letter (p, a, q) is such that $q \notin F$, then in the first right-to-left sweep (i.e., $i = 1$) the 2DFA halts and rejects.

The accept states are $(\lceil \log_2 n \rceil - 1, 0)$ and $(\lceil \log_2 n \rceil - 1, 1)$, if $\lceil \log_2 n \rceil$ is odd; in the even case a new accept state was introduced.

So far we assumed $n > 2$ (i.e., $\lceil \log_2 n \rceil \geq 2$). When $0 < n \leq 2$ then the previously mentioned n-state 1DFA recognizing L_0 can be used since, for $n \leq 2$, we have $n \leq 2\lceil \log_2 n \rceil + 1$.

Proof of (1). By (perhaps) replacing n by $n+1$, we may assume that the 1NFA N has a single accept state f with no out-edges (except if the empty word ε is accepted—this is dealt with separately later); so $F = \{f\}$. Now N has $\leq n+1$ states.

We think of Q as a subset of $\{0,1\}^k$, where $k = \lceil \log_2(n+1) \rceil$. Let us assume for now that $n > 2$, so $k > 1$.

The $\forall$-INFA for L_0 is based on the same idea as the sweeping 2DFA that we constructed earlier.

The set of *states* is $\{s_0, s_1\} \cup \{0, 1, \ldots, k-1\} \times \{0,1\}$. The *start* state is s_0; s_1 is a *sink* state. The set of *accept* states is $\{(i, \beta)/\text{the } i\text{th bit of f is } \beta\}$, where f is the unique accept state of N. If the empty word should be accepted, we make s_0 an accept state too. The state *transitions* of the $\forall$-1NFA are as follows (we write $s \cdot (p, a, q)$ to denote the set of states reached from state s, when a letter (p, a, q) is read):

$$s_0 \cdot (p, a, q) = \begin{cases} s_1 & \text{if } p \neq q_0 \quad (\text{where } q_0 \text{ is the start state of N}), \\ \{(j, \beta)/0 \leq j < k \text{ and the } j\text{th bit of q is } \beta\} & \text{otherwise.} \end{cases}$$

The only nondeterministic state (of $\forall$-type) is s_0; all other states are deterministic.

$$s_1 \cdot (p, a, q) = s_1;$$

$$(i, \beta_1) \cdot (p, a, q) = \begin{cases} s_1 & \text{if the } i\text{th bit of p is not } \beta_1, \\ (i, i\text{th bit of q}) & \text{otherwise.} \end{cases}$$

The easiest way to understand this $\forall$-1NFA, and to check that it recognizes L_0, is to apply the subset construction to it, and thus obtain an equivalent 1DFA. If a string $w \in \Delta^*$ does not start with $(q_0, \cdot, \cdot)$, this 1DFA will go to state $\{s_1\}$, and stay in it. After reading an initial segment... $(\cdot, \cdot, q)$ of w, the state of the 1DFA will either contain s_1 or it will be $\{(i, \beta_1)/0 \leq i < k$, the ith bit of q is $\beta_1\}$. If the next letter does not start with q, the next state will contain s_1; if the next letter is of the form (q, b, r) the next state will be $\{(j, \beta_2)/0 \leq j < k$, the jth bit of r is $\beta_2\}$. Finally when $(\cdot, \cdot, f)$ is encountered and s_1 has not been reached so far, the state will become $\{(j, \beta)/\text{the } j\text{th bit of f is } \beta\}$, which is an accept state of the 1DFA. Once s_1 is reached, the future states of the 1DFA will always contain s_1, and thus cannot be accept states. (Note that in the subset construction for a $\forall$1NFA, the accept states of the 1DFA are those sets that contain *only* accept states of the $\forall$1NFA.) □

Remark. By using Theorem 2.1 we can prove the following result:

$$[n \text{ state } \mathbf{1NFA}] \subseteq \text{L.P.HOM}[\lceil \log_2 n \rceil \text{ state } \mathbf{1AFA}].$$

This is weaker than Theorem 4.1(1) in that it uses "1AFA" instead of "$\forall$-1NFA"; but only $\lceil \log_2 n \rceil$ states are used instead of $2\lceil \log_2(n+1) \rceil + 2$.

Proof of the Remark. If L is recognized by a 1NFA with n states, then L^{rev} has a 1NFA with $\leq n$ states (where we allow 1NFAs to have several start states). We

can write $L^{rev} = \varphi(L'_0)$, where φ is a length-preserving homomorphism, and L'_0 is a "local" language, recognized by 1DFA with n states. (See the beginning of the proof of Theorem 4.1, which also works for 1NFAs with several start states.) Then, by Theorem 2.1, L'^{rev}_0 is recognized by a 1AFA with $\lceil \log_2 n \rceil$ states. Therefore $L = \varphi(L'^{rev}_0) \in$ L.P.HOM[$\lceil \log_2 n \rceil$ state 1AFA]. □

Remark on the Conciseness of the Description in Theorem 4.1. The given n-state 1NFA $N = (Q, \Sigma, \delta, q_0, F)$ has $|\delta|$ edges, with $|\delta| \leq |\Sigma| \cdot n^2$ (where $n = |Q|$).

In Theorem 4.1 we write L (the language recognized by N) as $h(L_0)$, where $h: \Delta^* \to \Sigma^*$ is a length-preserving homomorphism, and $L_0 \subseteq \Delta^*$ is a "local" language which has a 1DFA with n states. The alphabet Δ has size $|\delta| \leq |\Sigma| \cdot n^2$.

In Theorem 4.1(2), the halting-&-sweeping 2DFA has $2\lceil \log_2 n \rceil + 1$ states and $(|\delta| + 2)(2\lceil \log_2 n \rceil + 1)$ edges. (The "+2" accounts for the endmarkers on the tape.) So, the 2DFA has $O(\log n)$ times as many transitions as the given 1NFA.

In Theorem 4.1(1), the ∀-1NFA has $2\lceil \log_2(n+1) \rceil + 2$ states and

$$|\delta|(2\lceil \log_2(n+1) \rceil + 2)$$

edges. In other words, the ∀-1NFA has $O(\log n)$ times as many transitions as the original 1NFA.

We next prove the analogue of Theorem 4.1, for **2NFAs**.

Theorem 4.2. *For every $n > 0$ we have:*

(1) [n state **2NFA**] ⊆ L.P.HOM[$8n + 3$ state **∀-1NFA**].
(2) [n state **2NFA**] ⊆ L.P.HOM[$8n + 1$ state **halting & sweeping 2DFA**].

The ∀-1NFAs *in* (1) *are very special: only their start state is nondeterministic (of* ∀*-type); all other states are deterministic* (so they are like the parallel intersection machines of [SS]).

By applying the subset construction for ∀-1NFAs, we obtain, as a consequence of Theorem 4.2(1), [n state 2NFA] ⊆ [2^{8n+3} state 1NFA]. Later (Theorem 4.5) we improve this to [n state 2NFA] ⊆ [$8^n + 2$ state 1NFA]. This improves the 2NFA-to-1NFA conversion of [HU] (see Theorem 2.3(2)) by replacing $n!(1 + \varepsilon(n))$ by $8^n + 2$.

Other consequences of Theorem 4.2 (and Theorem 4.1) are:

Corollary 4.3. *For all $n > 0$:*

(1) [n-state **2AFA**] ⊆ [2^{cn^2} state **1NFA**],
(2) [n-state **halting 2AFA**] ⊆ [2^{cn} state **1NFA**],
(3) [n-state **1-pebble 2NFA**] ⊆ [2^{cn^2} state **1NFA**],

where c *is a constant* (> 1).

Compare this with Fact 3.2 and Theorem 3.4.

Proof of Corollary 4.3(1). First we use Theorem 2.6, then we apply Theorem 4.2(1) to the 2DFA that appears; this yields L.P.HOM[cn^2 state $\forall$-1NFA]. Applying the subset construction to the $\forall$-1NFA we get L.P.HOM[2^{cn^2} state 1DFA], which is contained in [2^{cn^2} state 1NFA]. (Note that L.P.HOM[k state 1NFA] = [k state 1NFA].) Parts (2) and (3) of the corollary are proved in the same way. □

Corollary 4.4. *The four families* L.P.HOM[n state **2DFA**], L.P.HOM[n state $\forall$-**1NFA**], L.P.HOM[n state **halting-&-sweeping 2DFA**], *and* [$\mathbf{2^n}$ state **1NFA**], *are* **equivalent** *up to linear changes in the parameter n.*

The first of these families is exactly characterized by a machine model in [B3].

Proof. L.P.HOM[n state halting-&-sweeping 2DFA] $\subseteq$ L.P.HOM[n state 2DFA] $\subseteq$ L.P.HOM[$8n + 3$ state $\forall$-1NFA], by Theorem 4.2(1). Also, by applying the subset construction for $\forall$-1NFAs, we get

L.P.HOM[n state $\forall$-1NFA] $\subseteq$ L.P.HOM[2^n state 1DFA] $\subseteq$ [2^n state 1NFA].

Finally, [2^n state 1NFA] $\subseteq$ L.P.HOM[$2n + 1$ state halting-&-sweeping 2DFA], by Theorem 4.1(2). □

A last remark about Theorem 4.2(2), before we prove Theorem 4.2: we already knew from Fact 2.5 that [n state 2NFA] $\subseteq$ L.P.HOM[n state 2DFA]; Theorem 4.2(2) says that, in combination with length-preserving homomorphisms, sweeping-and-halting 2DFAs are as powerful as general 2DFAs (up to a linear increase in the number of states).

Proof of Theorem 4.2. Let $L \subseteq \Sigma^*$ be recognized by a 2NFA $N = (Q, \Sigma, \delta, q_0, F)$ with $Q = \{0, \ldots, n-1\}$; the tape has a left (and a right) endmarker $\langle$ (resp. $\rangle$); the transition relation $\delta: Q \times \Sigma \to Q \times \{-1, +1\}$ indicates the possible next states and directions of movement of the reading head; see the Appendix. The proof scheme is similar to the one for Theorem 4.1. We give a language L_0 over some alphabet Δ, and a length-preserving homomorphism $h: \Delta^* \to \Sigma^*$ such that $L = h(L_0)$; we have the additional property that L_0 is *strictly locally 3-testable* in the (restricted) sense that $L_0 = D \cup (A\Delta^+ \cap \Delta^+B - \Delta^*C\Delta^*)$, for some sets A, B, $D \subseteq \Delta$ and $C \subseteq \Delta^3$. (For more information on strictly locally testable languages, see [MP]; see also [B5] for other applications of strict local testability.) Finally, in order to prove Theorem 4.2 we show that L_0 can be recognized by a $\forall$-1NFA with $8n + 3$ states, and by a halting-&-sweeping 2DFA with $8n + 1$ states. Let us first construct Δ, L_0, and h.

Every letter $a \in \Sigma$ induces the following binary relations on the set of states:

the *left relation* $\overleftarrow{a} = \{(q_1, q_2) \in Q \times Q / (q_2, -1) \in \delta(q_1, a)\}$;
the *right relation* $\overrightarrow{a} = \{(p_1, p_2) \in Q \times Q / (p_2, +1) \in \delta(p_1, a)\}$;

at the ends of the tape the letter a induces the relations

$[\langle a \leftrightarrows] = \{(p_1, p_2) \in Q \times Q / \langle p_1 a \vdash^* \langle a p_2\}$

and

$$[\rightleftarrows a\rangle] = \{(q_1, q_2) \in Q \times Q / q_1 a\rangle \vdash^* q a\rangle \text{ and } x q a\rangle \vdash q_2 x a\rangle, \text{ for some q in Q and } x \text{ in } \Sigma\}$$

(see Section A3 of the Appendix or [HU] for the definition of $\vdash^*$; the Appendix also discusses $[\cdots \leftrightarrows]$ and $[\rightleftarrows \cdots]$ in more detail).

The 2NFA starts its computation on input $w \in \Sigma^*$ in the configuration $q_0\langle w\rangle$ (i.e., it scans the left endmarker); it *accepts* when it makes a transition $\langle wq\rangle \vdash \langle w\rangle q_F$ for some $q_F \in F$, $q \in Q$.

The alphabet Δ is now defined as follows:

$$\Delta = \{f, a, g)/a \in \Sigma, f \subseteq \hat{a}, g \subseteq \hat{a}, \text{ f and g are } \textit{injective} \text{ partial } \textit{functions} \text{ from Q to Q, } \textit{Dom } g \neq \varnothing \text{ and } \textit{Dom } f \cap \textit{Dom } g = \varnothing\} \cup \Delta_L \cup \Delta_R \cup (\Sigma \cap L),$$

where

$$\Delta_L = \{(\varnothing, a, g)/a \in \Sigma, g \subseteq [\langle a \leftrightarrows], \text{ g is an } \textit{injective} \text{ partial } \textit{function}, \text{ and there is a state q in } \textit{Dom } g \text{ such that } q_0\langle \vdash^* \langle q\},$$

$$\Delta_R = \{(f, a, \varnothing)/a \in \Sigma, f \subseteq [\rightleftarrows a\rangle], \text{ f is an } \textit{injective} \text{ partial } \textit{function}, \text{ and there is a state q } \textit{not} \text{ in } \textit{Dom } f \text{ such that } q a\rangle \vdash^* a\rangle q_F \text{ for some } q_F \in F\}.$$

Terminology: A function f from Q to Q is partial if its domain *Dom* f is a (possibly strict) subset of Q; for two relations R_1, R_2 we write $R_1 \subseteq R_2$ if the graph of $R_1 \subseteq Q \times Q$ is a subset of the graph of R_2.

The homomorphism h is defined by $h(f, a, g) = h(\varnothing, a, g) = h(f, a, \varnothing) = h(a) = a$.

Intuitively, (f, *a*, g) consists of a letter $a \in \Sigma$ together with a choice of transitions on *a* to be used in an accepting computation; g corresponds to the chosen left-to-right moves, f to the chosen right-to-left moves; the special properties required for f and g correspond to the fact that we may assume that an accepting computation of a 2NFA is a *path* in the computation graph; there are no branchings ($\cdot <$, $> \cdot$, $\leftarrow \cdot \rightarrow$) and no confluences ($> \cdot$, $\cdot <$, $\rightarrow \cdot \leftarrow$) on a path. (Note: A path is a walk in which no vertex is repeated.) See Appendix A3 for the definition of "computation graph" and "accepting computation."

The locally testable language L_0 should describe accepting paths of the 2NFA for all possible accepted inputs. Since a word in L_0 should describe a *path*, we must rule out *branchings* (in fact, all cases of branching are already ruled out by the definition of the alphabet Δ); and we must rule out *confluence* (two cases of confluence are ruled out by the definition of Δ, a third case is of the form $\rightarrow \cdot \leftarrow$). We must also rule out *sinks* and *sources* along the path (these are vertices where the path would end or start inside the word). Finally at the left end of the input we must enforce that the computation of the 2NFA starts in the start state q_0 and handles the left endmarker correctly; similarly, at the right end we must deal with the accept states and with the right endmarker. All these conditions on a

computation path are *local.* The formal definition of L_0 is as follows:

$L_0 = \{w \in \Delta^* / w$ satisfies conditions (1)–(5) given below$\}$.

(1) (*No sink*) For every subsegment $(f_1, a_1, g_1)(f_2, a_2, g_2)$ of w:

Range $g_1 \subseteq$ *Dom* $f_2 \cup$ *Dom* g_2,

Range $f_2 \subseteq$ *Dom* $f_1 \cup$ *Dom* g_1.

(2) (*No source*) For every subsegment $(f_1, a_1, g_1)(f_2, a_2, g_2)(f_3, a_3, g_3)$ of w:

Dom $f_2 \cup$ *Dom* $g_2 \subseteq$ *Range* $g_1 \cup$ *Range* f_3.

By condition (1), this "$\subseteq$" is actually an equality.

(3) (*No confluence*) For every subsegment $(f_1, a_1, g_1)(f_2, a_2, g_2)(f_3, a_3, g_3)$ of w:

Range $g_1 \cap$ *Range* $f_3 = \varnothing$.

(4) (*Left and right end*) If $|w| \geq 2$, then the leftmost (resp. rightmost) letter of w belongs to Δ_L (resp. Δ_R).

(5) If $|w| \leq 1$, then $w \in D$, where $D = (\Sigma \cup \{\varepsilon\}) \cap L$.

Lemma. *For* Δ, h, L_0 *just constructed we have* $L = h(L_0)$.

Proof of the Lemma. It is easy to see that if $y \in \Sigma^*$ is accepted by the 2NFA, then any loop-free accepting computation (path) of y can be described by a word $x \in L_0$, and we have $y = h(x)$. Conversely, suppose x belongs to L_0. Because of the local conditions (1)–(5) and the restrictions on the alphabet Δ, x must describe an accepting path of the 2NFA on input h(x), so $h(x) \in L$.

One subtlety should be mentioned: Although $x \in L_0$ describes exactly one accepting path of the 2NFA on input h(x), the word x (in addition) may also describe some cycles, not connected to the path; see Figure 1. These cycles have no influence on the acceptance of h(x) by the 2NFA. The local conditions (1)–(5) and the definition of the alphabet cannot prevent such cycles. □

Proof of (1) of Theorem 4.2. We construct a $\forall$-1NFA recognizing L_0; the idea is similar to the previous construction of a $\forall$-1NFA for Theorem 4.1(1).

The set of *states* is

$$\{s_0, s_1, s_F\} \cup \bigcup_{0 \leq i \leq n-1} \{\rightarrow, \not\rightarrow\} \times \{\uparrow\rightarrow, \uparrow\not\rightarrow, \updownarrow\rightarrow, \updownarrow\not\rightarrow\} \times \{i\}.$$

The intuitive meaning of a state

$$(d_2, d_1, i) \in \{\rightarrow, \not\rightarrow\} \times \{\uparrow\rightarrow, \uparrow\not\rightarrow, \updownarrow\rightarrow, \updownarrow\not\rightarrow\} \times \{i\}$$

is as follows: Suppose the $\forall$-1NFA is currently in configuration

$$\cdots \alpha_{j-2}\alpha_{j-1}(d_2, d_1, i)\alpha_j \cdots;$$

the current state is (d_2, d_1, i), and the reading head is under α_j. In this state, $d_2 = \rightarrow$ iff α_{j-2} is of the form $\alpha_{j-2} = (f_{j-2}, a_{j-2}, g_{j-2}) \in \Delta$ with $i \in$ *Range* g_{j-2};

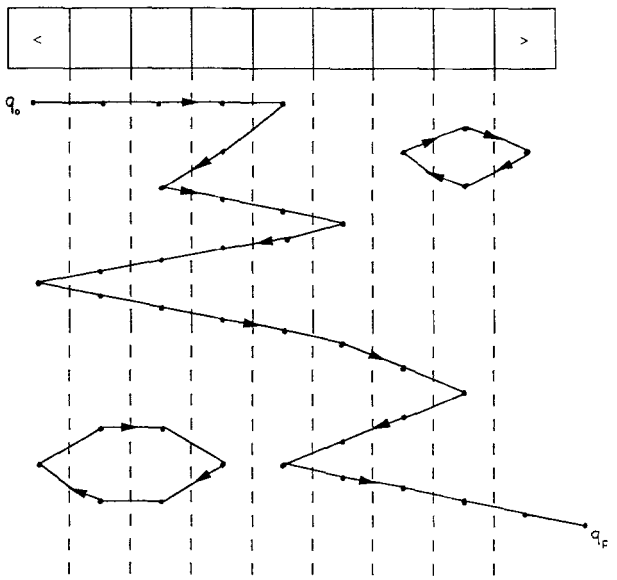

Fig. 1. Example of a path and isolated cycles described by a word in the local language L_0 of a 2NFA.

$d_2 = \not\rightarrow$ otherwise. (In words: d_2 remembers whether, in the path described by a word $w \in \Delta^*$, an edge points from the left to vertex i at position $j-1$; the current position is j.)

Let $\alpha_{j-1} = (f_{j-1}, a_{j-1}, g_{j-1}) \in \Delta$; the coordinate d_1 of the above state is as follows:

$d_1 = \uparrow\rightarrow$ iff $i \in Dom\ f_{j-1} \cup Dom\ g_{j-1}$, and $i \in Range\ g_{j-1}$;
$d_1 = \uparrow\not\rightarrow$ iff $i \in Dom\ f_{j-1} \cup Dom\ g_{j-1}$, and $i \notin Range\ g_{j-1}$;
$d_1 = \updownarrow\rightarrow$ iff $i \notin Dom\ f_{j-1} \cup Dom\ g_{j-1}$, and $i \in Range\ g_{j-1}$;
$d_1 = \updownarrow\not\rightarrow$ iff $i \notin Dom\ f_{j-1} \cup Dom\ g_{j-1}$, and $i \notin Range\ g_{j-1}$.

So d_1 remembers whether, in the path described by a word $w \in \Delta^*$, an edge points out of vertex i at position $j-1$ and whether an edge points from the left to vertex i at the current position j. We see below that the state contains enough information so that, together with the input letter α_j, it determines the next state and it determines whether the three rules (1), (2), (3) defining L_0 are violated in vertex i at positions $j-1$ or j.

The *start* state is s_0, and s_1 is a sink state. The set of *accept* states is $\{s_F\}$; if the empty word should be accepted, s_0 is an accept state too. Formally, the state *transitions* of the $\forall$-1NFA are as follows:

$$s_0 \cdot (f, a, g) = \begin{cases} s_1 & \text{if } (f, a, g) \notin \Delta_L, \\ \{(d_2, d_1, i)/0 \leq i < n-1, \text{and } d_2 \text{ and } d_1 \text{ are obtained from g as given below}\} & \text{otherwise;} \end{cases}$$

here $d_2 = \rightarrow$ if $q_0 \langle \stackrel{*}{\vdash} \langle i$, $d_2 = \not\rightarrow$ otherwise; and

$$d_1 = \begin{cases} \uparrow\rightarrow & \text{if } i \in \mathit{Range}\, g \cap \mathit{Dom}\, g, \\ \updownarrow\rightarrow & \text{if } i \in \mathit{Range}\, g - \mathit{Dom}\, g, \\ \uparrow\not\rightarrow & \text{if } i \in \mathit{Dom}\, g - \mathit{Range}\, g, \\ \updownarrow\not\rightarrow & \text{if } i \notin \mathit{Dom}\, g \cup \mathit{Range}\, g. \end{cases}$$

So the start state is nondeterministic (in the $\forall$-sense). All other states are deterministic. For $a \in D - \{\varepsilon\}$:

$$s_0 \cdot a = s_F.$$

For all $x \in \Delta$:

$$s_1 \cdot x = s_1 = s_F \cdot x.$$

For a state (d_2, d_1, i) (other than s_0, s_1, s_F), and for $x \in \Delta_L \cup (D - \{\varepsilon\})$:

$$(d_2, d_1, i) \cdot x = s_1.$$

For a state (d_2, d_1, i) (other than s_0, s_1, s_F), and for an input

$(f, a, g) \in \Delta - (\Delta_L \cup (D - \{\varepsilon\}) \cup \Delta_R)$:

$$(d_2, d_1, i) \cdot (f, a, g) = \begin{cases} s_1 & \text{if condition (1) or (2) or (3) is violated at } i, \\ (d_2', d_1', i) & \text{otherwise,} \end{cases}$$

where d_2' is obtained from d_1 by dropping the out-edge information $\uparrow$ or $\updownarrow$, and where

$$d_1' = \begin{cases} \uparrow\rightarrow & \text{if } i \in \mathit{Range}\, g \cap (\mathit{Dom}\, f \cup \mathit{Dom}\, g) \\ \updownarrow\rightarrow & \text{if } i \in \mathit{Range}\, g - (\mathit{Dom}\, f \cup \mathit{Dom}\, g), \\ \uparrow\not\rightarrow & \text{if } i \in (\mathit{Dom}\, f \cup \mathit{Dom}\, g) - \mathit{Range}\, g, \\ \updownarrow\not\rightarrow & \text{if } i \notin \mathit{Dom}\, f \cup \mathit{Dom}\, g \cup \mathit{Range}\, g. \end{cases}$$

Suppose $\forall$-1NFA is currently in configuration $\cdots \alpha_{j-2}\alpha_{j-1}(d_2, d_1, i)\alpha_j \cdots$; the reading head is now under $\alpha_j = (f, a, g) \in \Delta$. Condition (1) is violated if $d_1 \in \{\uparrow\rightarrow, \updownarrow\rightarrow\}$ but $i \notin \mathit{Dom}\, f \cup \mathit{Dom}\, g$ (there is an edge pointing from the left to vertex i at postion j but there is no edge out of this vertex; so i at position j is a *sink*), or if $i \in \mathit{Range}\, f$ but $d_1 \in \{\updownarrow\rightarrow, \updownarrow\not\rightarrow\}$ (there is an edge from the right into i at position $j - 1$, but there is no edge out of i; then i is a *sink* at position $j - 1$).

Condition (2) is violated if $d_1 \in \{\uparrow\rightarrow, \uparrow\not\rightarrow\}$ (there is an edge out of vertex i at position $j - 1$), but $i \notin \mathit{Range}\, f$ (no edge goes into i from letter α_j on the right) and $d_2 = \not\rightarrow$ (no edge goes into i from the left); then vertex i at position $j - 1$ is a *source*.

Condition (3) is violated if $d_2 = \rightarrow$ (and edge points from the left to vertex i at position $j - 1$) and $i \in \mathit{Range}\, f$ (an edge points to i at position $j - 1$ from the right); then there is *confluence* into i at position $j - 1$.

Finally, for a state (d_2, d_1, i) (other than s_0, s_1, s_F), and for an input $(f, a, \varnothing) \in \Delta_R$:

$$(d_2, d_1, i) \cdot (f, a, \varnothing) = \begin{cases} s_1 & \text{if condition (1) or (2) or (3) is violated at } i, \\ s_F & \text{otherwise.} \end{cases}$$

The violation of conditions (1)–(3) is determined as before (except that now $g = \varnothing$, which makes $Dom\ g = \varnothing$).

Proof of (2) of Theorem 4.2. We construct a sweeping and halting 2DFA recognizing L_0. This 2DFA, on input $w \in \Delta^*$, makes n sweeps (or $n+1$ if n is even, in order to make the head end up at the right end of the tape in accepting computations); n is the number of states of the original 2NFA. The sweeps are numbered from 0 to $n-1$ (or n); even-numbered sweeps go from left to right, odd-numbered sweeps go from right to left. At the beginning of sweep number 0 the sweeping 2DFA checks whether the input is in D or begins with a letter in Δ_L; and at the end of sweep 0 it checks that the input's rightmost letter is in Δ_R. The main part of the computation of the 2DFA consists in checking whether the local conditions (1)–(3) (in the definition of L_0) hold everywhere in the input.

Suppose that the set of states of the original 2NFA is $Q = \{0, \ldots, n-1\}$. In sweep number i ($0 \leq i \leq n-1$) the 2DFA checks whether conditions (1)–(3) hold everywhere for state i; that is, everywhere in the input, i is not a sink nor a source, and we do not have confluence toward i. To check all this, the sweeping 2DFA uses the following set of eight states in sweep number i:

$$\{\rightarrow, \not\rightarrow\} \times \{\uparrow\rightarrow, \uparrow\not\rightarrow, \updownarrow\rightarrow, \updownarrow\not\rightarrow\} \times \{i\},$$

when i is even (i.e., the sweep goes from left to right). The meaning of these states and the state-transitions in this sweep are as follows: Suppose the 2DFA is in state $(d_2, d_1, i) \in \{\rightarrow, \not\rightarrow\} \times \{\uparrow\rightarrow, \uparrow\not\rightarrow, \updownarrow\rightarrow, \updownarrow\not\rightarrow\} \times \{i\}$, and it is currently in configuration $\cdots \alpha_{j-2}\alpha_{j-1}(d_2, d_1, i)\alpha_j \cdots$ (during the left-to-right sweep i); the reading head is now under α_j. In this state, $d_2 = \rightarrow$ iff α_{j-2} is of the form

$$\alpha_{j-2} = (f_{j-2}, a_{j-2}, g_{j-2}) \in \Delta$$

with $i \in Range\ g_{j-2}$; $d_2 = \not\rightarrow$ otherwise. (In words: d_2 remembers whether in the path described by a word $w \in \Delta^*$ an edge points from the left to vertex i at position $j-1$; the current position is j.) Let $\alpha_{j-1} = (f_{j-1}, a_{j-1}, g_{j-1}) \in \Delta$; the coordinate d_1 of the above state has already been described in detail at the beginning of the proof of (1) of Theorem 4.2. Intuitively, d_1 remembers whether an edge points out of vertex i at position $j-1$ and whether an edge points from the left to vertex i at the current position j.

When i is odd (right-to-left sweep) the states used are the set

$$\{i\} \times \{\not\leftarrow\uparrow, \leftarrow\updownarrow, \not\leftarrow\uparrow, \leftarrow\uparrow\} \times \{\not\leftarrow, \leftarrow\};$$

the meaning is symmetric to the case when i is even.

During each sweep the 2DFA acts like a classical window automaton (as used for strictly locally testable languages, see [MP]). The turns (at the ends of the sweeps) are handled in the same way as in the proof of (2) of Theorem 4.1. So far we have used $8n$ states, namely the set

$$\bigcup_{0 \leq i \leq n-1} (\{\rightarrow, \not\rightarrow\} \times \{\uparrow\rightarrow, \uparrow\not\rightarrow, \updownarrow\rightarrow, \updownarrow\not\rightarrow\} \times \{i\}$$

$$\cup \{i\} \times \{\not\leftarrow\updownarrow, \leftarrow\updownarrow, \not\leftarrow\uparrow, \leftarrow\uparrow\} \times \{\not\leftarrow, \leftarrow\}).$$

The start state is $(\not\to, \updownarrow\not\to, 0)$. We use an additional (new) state as accept state. If n is even this state is also used to make accepting computations of the 2DFA end at the right end of the input. So the number of states of the 2DFA is $8n + 1$.

It can be seen (in the same way as in the proof of (1) of this theorem) that the information in the state, together with the currently read input letter α_j, is sufficient to check rules (1)–(3), and to determine the next state. The 2DFA halts (in a nonaccept state) as soon as it finds out that one of the rules is violated. This completes the proof of Theorem 4.2. □

Theorem 4.5. *For every $n > 0$ we have*

$$[n \text{ state } \mathbf{2NFA}] \subseteq [8^n + 2 \text{ state } \mathbf{1NFA}].$$

Comments. Compare this with Theorem 5.1 (to convert an n-state 2DFA into an equivalent 1NFA we need $\geq \sqrt{2^{n-1}}$ states in general); so, up to a linear change in the parameter n, Theorem 4.5 is optimal. Compare Theorem 4.5 also with Corollary 2.4 (which says that an n-state 2NFA can be simulated by a 1AFA with n^2 states), and with Theorem 2.3(3).

Proof of Theorem 4.5. We apply the *reachable* subset construction to the $\forall$-1NFA of Theorem 4.2(1), in order to obtain a partial 1DFA recognizing L_0. The state-sets we obtain that way are: $\{s_0\}$ (for the start state), the accepting set $\{s_F\}$, and sets of the form $P = \{(d_2^{(i)}, d_1^{(i)}, i)/0 \leq i < n\}$ with

$$d_2^{(i)} \in \{\to, \not\to\},\ d_1^{(i)} \in \{\uparrow\to, \uparrow\not\to, \updownarrow\to, \uparrow\not\to\}.$$

Indeed we have

Claim. *Each such set* P *has the special property that for each i, one and only one value of $(d_2^{(i)}, d_1^{(i)})$ appears in* P *(e.g., if $(\to, \uparrow\to, i) \in P$, then for this i no other (d_2, d_1, i) is in* P*).*

Proof of the Claim (by induction on the length of the computation that leads to P). The claim is true at the beginning of the computation; see the definition of the set $s_0 \cdot (f, a, g)$ in the construction of the $\forall$-1NFA. Suppose that P is of the form $\{(d_2^{(i)}, d_1^{(i)}, i)/0 \leq i < n\}$, as claimed, and that a letter (f, a, g) is read. Then the next set is $P \cdot (f, a, g) = \{(d_2^{(i)}, d_1^{(i)}, i) \cdot (f, a, g)/0 \leq i < n\} = \{(d_2'^{(i)}, d_1'^{(i)}, i)/0 \leq i < n\}$, where $d_2'^{(i)}$ and $d_1'^{(i)}$ are uniquely determined according to the state-transitions of the $\forall$-1NFA. Since there was exactly one $(d_2^{(i)}, d_1^{(i)})$ for i, there will also be exactly one $(d_2'^{(i)}, d_1'^{(i)})$ for i. This proves the claim. □

No other sets are needed; the next-state set will be made undefined when the sink-state s_1 is reached in the $\forall$-1NFA. Thus we obtain a partial 1DFA with $8^n + 2$ states, recognizing L_0. Finally, since $L = h(L_0)$ (where h is a length-preserving homomorphism), we obtain a 1NFA with $8^n + 2$ states recognizing L (by replacing each edge labeled, say $x \in \Delta$, in the DFA by an edge labeled $h(x) \in \Sigma$). □

Remark. If instead of the classical model of a 2NFA the 2NFAs as in [B1] or [B2] are used (where the head stays between tape cells in a given configuration and moves over a cell in a transition), the corresponding local language L_0 is a little simpler: it is 2-testable (instead of 3), and it is recognized by a $\forall$-1NFA (where only the start state is nondeterministic) with $4n + O(1)$ states, and by a halting-&-sweeping 2DFA with $4n + O(1)$ states. Also, such a 2NFA with n states can be simulated by a 1NFA with $4^n + O(1)$ states. Both models of 2NFA are equivalent (up to doubling the number of states).

5. New Results on Incompressibility, and Lower Bounds

Theorem 5.1. *For all $n > 0$ we have:*

If [n state **2DFA**] $\subseteq$ [f(n) state **1NFA**], *then* $f(n) \geq \sqrt{2^{n-1}}$.
If [n state **2DFA**] $\subseteq$ [g(n) state **1NFA**] $\cup$ [g(n) state $\forall$-**1NFA**], *then* $g(n) \geq \sqrt{2^{n-3}}$.

These results are still true when **2DFA** *is replaced by* **halting-&-sweeping 2DFA**.

Comments. This improves the results of Sakoda and Sipser [SS]; see Theorem 3.3. Also, compare the incompressibility result, Theorem 5.1, with the inclusion results, Theorem 2.3(3) due to Vardi [V] and Theorem 4.5.

Proof of Theorem 5.1. Suppose

[n state sweeping-&-halting 2DFA] $\subseteq$ [f(n) state 1NFA];

applying L.P.HOM to this inclusion we obtain

L.P.HOM[n state sweeping-&-halting 2DFA]
$\subseteq$ L.P.HOM[f(n) state 1NFA] (=[f(n) state 1NFA]).

By Theorem 4.1(2),

$[2^{(n-1)/2}$ state 1NFA] $\subseteq$ L.P.HOM[n state sweeping-&-halting 2DFA].

Together these incusions imply

$[2^{(n-1)/2}$ state 1NFA] $\subseteq$ [f(n) state 1NFA].

Thus $2^{(n-1)/2} \leq f(n)$.

Moreover:

co-[n state sweeping-&-halting 2DFA]
$\subseteq$ [$n + 2$ state sweeping-&-halting 2DFA],

where *co-* is the complementation operator (defined in Section 1). (Proof of this inclusion: If L is recognized by a sweeping and halting 2DFA **A** then $\bar{L}$ is recognized by the sweeping and halting 2DFA **A′** defined as follows: The states of **A′** are the states of **A** together with two new states s and f, where f is the only

accept state of $\mathbf{A}'$. $\mathbf{A}'$ simulates $\mathbf{A}$, but when a computation of $\mathbf{A}$ halts and rejects at a position other than the right end of the tape, during a right-to-left sweep, then $\mathbf{A}'$ goes to state s and finishes the sweep, then goes to state f and performs a left-to-right sweep. If $\mathbf{A}$ halted during a left-to-right sweep, $\mathbf{A}'$ goes to state f and finishes the sweep.) Thus, if

$$[n + 2 \text{ state sweeping-\&-halting 2DFA}] \subseteq [g(n + 2) \text{ state } \forall\text{-1NFA}],$$

then

$$\begin{aligned}&[n \text{ state sweeping-\&-halting 2DFA}]\\ &\quad \subseteq \textit{co}\text{-}[n + 2 \text{ state sweeping-\&-halting 2DFA}]\\ &\quad \subseteq \textit{co}\text{-}[g(n + 2) \textit{ state } \forall\text{-1NFA}]\\ &\quad = [g(n + 2) \text{ state 1NFA}].\end{aligned}$$

Now by the first part of this theorem, we obtain $g(n + 2) \geq \sqrt{2^{n-1}}$. □

Theorem 5.2. *For all $n > 0$ we have*:

(1) *If* $[n$ state **2DFA**$] \subseteq [f(n)$ state **2AFA**$]$, *then* $f(n) \geq c \cdot \sqrt{n}$ (*for some constant* $c > 0$).

The result is also true if "2DFA" *is replaced by* **sweeping-and-halting 2DFA**.

(2) *If* $[n$ state **1NFA**$] \subseteq [g(n)$ state **2AFA**$]$, *then* $g(n) \geq d \cdot \sqrt{n}$ (*for some constant* $d > 0$).

The result is also true when "1NFA" *is replaced by* $\forall$-**1NFA** *or, a fortiori, by* **1AFA**.

(3) *Statements* (1) *and* (2) *remain true if* "2AFA" *is replaced by* **1-pebble 2NFA**:

If $[n$ state **2DFA**$] \subseteq [f(n)$ state **1-pebble 2NFA**$]$, *then* $f(n) \geq c \cdot \sqrt{n}$ (*for some constant* $c > 0$).

If $[n$ state **1NFA**$] \subseteq [g(n)$ state **1-pebble 2NFA**$]$, *then* $g(n) \geq d \cdot \sqrt{n}$ (*for some constant* $d > 0$).

(4) *Statements* (1) *and* (2) *remain true when* "2AFA" *is replaced by* **halting 2AFA**, *and* $\sqrt{n}$ *is replaced by* n:

If $[n$ state **2DFA**$] \subseteq [f(n)$ state **halting 2AFA**$]$, *then* $f(n) \geq c \cdot n$ (*for some constant* $c > 0$).

If $[n$ state **1NFA**$] \subseteq [g(n)$ state **halting 2AFA**$]$, *then* $g(n) \geq d \cdot n$ (*for some constant* $d > 0$).

Comments. According to this result, 2DFAs and 1NFAs cannot be compressed much by 2AFAs; but (by Corollary 2.2) 1DFAs can be compressed logarithmically by 2AFAs.

Proof. (1) $[n$ state 2DFA$] \subseteq [f(n)$ state 2AFA$] \subseteq$ (by Theorem 2.6)

$$\text{L.P.HOM}[c' \cdot (f(n))^2 \text{ state 2DFA}] \subseteq \text{(by Theorem 4.5) } [2^{d' \cdot (f(n))^2} \text{ state 1NFA}]$$

for some constants c', d' > 1. Now, by Theorem 5.1 (which also holds if the 2DFA is halting-&-sweeping), $2^{d' \cdot (f(n))^2} \geq 2^{(n-1)/2}$, and Theorem 5.2(1) follows from this.

(2) Since [n state 2AFA] is closed under the complementation operator ***co-*** we have

[n state **1NFA**] $\subseteq$ [g(n) state 2AFA] iff [n state $\forall$-1NFA] $\subseteq$ [g(n) state 2AFA]

Moreover,

$$\begin{aligned}[n \text{ state } \forall\text{-1NFA}] &\subseteq [g(n) \text{ state 2AFA}] \\ &\subseteq \text{(by Theorem 2.6) L.P.HOM}[c' \cdot (g(n))^2 \text{ state 2DFA}] \\ &\subseteq \text{(by Theorem 4.5) } [2^{d' \cdot (g(n))^2} \text{ state 1NFA}]\end{aligned}$$

for some constants c', d' > 1. So [n state $\forall$-1NFA] $\subseteq$ [$2^{d' \cdot (g(n))^2}$ state 1NFA]; thus, by Theorem 3.4 [SS], $2^{d' \cdot (g(n))^2} \geq 2^n$.

(3) This condition uses literally the same proof as (1) and (2), with "2AFA" replaced by "1-pebble 2NFA."

(4) This condition is also proved in exactly the same way as (1) and (2), with "2AFA" replaced by "halting 2AFA," "$(f(n))^2$" replaced by "$f(n)$," and "$(g(n))^2$" replaced by "$g(n)$." □

Corollary 5.3. *For all $n \geq 1$ we have*

If L.P.HOM[n state **2DFA**] $\subseteq$ [f(n) state **2AFA**], *then* $f(n) \geq c2^{n/4}$ (*for some constant* $c > 0$).

The same is true if "2AFA" *is replaced by* **1-pebble 2NFA**.

Proof. By Theorem 4.1 we have

$$[2^{(n-1)/2} \text{ state 1NFA}] \subseteq \text{L.P.HOM}[n \text{ state 2DFA}] \subseteq [f(n) \text{ state 2AFA}].$$

By Theorem 5.2(2), $f(n) \geq d \cdot \sqrt{2^{(n-1)/2}} \geq c \cdot 2^{n/4}$. For 1-pebble 2NFAs we use Theorem 5.2(3). □

A consequence of Corollary 5.3 is that n-state *nondeterministic Hennie machines*, studied in [B3], are exponentially more powerful than 2AFAs. Hennie machines [H] are nondeterministic single-tape n-state Turing machines which never leave the input portion of the tape, and which never visit the same tape-cell in the same state (during some accepting computation, on each accepted input). In fact we have the stronger result that nondeterministic Hennie machines are exponentially more powerful than deterministic ones:

Corollary 5.4. *For all $n \geq 1$ we have*

If [n state **nondet. H.M.**] $\subseteq$ [f(n) state **det. H.M.**], *then* $f(n) > \sqrt{2^{n-5}}$.

Remarks. "H.M." stands for Hennie machine, as just defined, "det" stands for deterministic. In [B3] it was proved that

L.P.HOM[n state 2DFA] = [n state nondet. H.M.].

Proof of Corollary 5.4. By the remark and by Theorem 4.1 we have

$$[2^{(n-1)/2} \text{ state 1NFA}] \subseteq [n \text{ state nondet. H.M.}] \subseteq [f(n) \text{ state det. H.M.}].$$

However, the latter class is closed under complement (see Section 4 of [B3]), so we have

$$\begin{aligned}[2^{(n-1)/2} \text{ state } \forall\text{-1NFA}] &\subseteq [f(n) \text{ state det. H.M.}] \\ &\subseteq (\text{by the above remark}) \text{ L.P.HOM}[f(n) \text{ state 2DFA}] \\ &\subseteq (\text{by Theorem 4.5}) [8^{f(n)} + 2 \text{ state 1NFA}].\end{aligned}$$

Then (by Theorem 3.4) $8^{f(n)} + 2 \geq 2^{2^{(n-1)/2}}$. Corollary 5.4 then follows. □

In exactly the same way we prove

If [n state non-det. H.M.] $\subseteq$ **co**-[$f(n)$ state nondet. H.M.], *then* $f(n) > \sqrt{2^{n-5}}$.

Corollary 5.5. *For all* $n \geq 1$:

(a) *If* [n state $\forall$-1NFA] $\subseteq$ L.P.HOM[$f(n)$ state $\forall$-1NFA], *then* $f(n) \geq n$.
(b) *If* L.P.HOM[n state $\forall$-1NFA] $\subseteq$ [$g(n)$ state $\forall$-1NFA], *then* $g(n) > 2^{2^{(n-3)/2}}$.

This shows how "differently shaped" the classes [n state $\forall$-1NFA] and L.P.HOM[m state $\forall$-1NFA] are: to fit the first into the second, no compression is possible ($m \geq n$); to fit the second into the first, m grows by a double exponential

Proof. (a) [n state $\forall$1NFA] $\subseteq$ L.P.HOM[$f(n)$ state $\forall$-1NFA] $\subseteq$ [$2^{f(n)}$ state 1NFA] (by the subset construction). Then, by Theorem 3.4 [SS], $2^{f(n)} \geq 2^n$, and thus $f(n) \geq n$.

(b) By Theorem 4.1(1), [$2^{(n-2)/2} - 1$ state 1NFA] $\subseteq$ L.P.HOM[n state $\forall$-1NFA], and the latter class is contained in [$g(n)$ state $\forall$-1NFA] by assumption. So [$2^{(n-2)/2} - 1$ state 1NFA] $\subseteq$ [$g(n)$ state $\forall$-1NFA]. Now, by Theorem 3.4, $g(n) \geq 2^{(2^{(n-2)/2}-1)}$. □

The next theorem does not add significantly to the previous results of this section, but we include it because it uses a very different proof scheme.

Theorem 5.6. *If, for all* n, [n state 2DFA] $\subseteq$ L.P.HOM[$f(n)$ state 2DFA], *then*:

(a) $f(n) \geq (n-2)/6$ *for all* n,
(b) $f(n) \geq n$ *for infinitely many* n.

Proof. For (a) we use the previous techniques:

$$\begin{aligned}[n \text{ state 2DFA}] &\subseteq \text{L.P.HOM}[f(n) \text{ state 2DFA}] \\ &\subseteq (\text{by Theorem 4.5}) [8^{f(n)} + 2 \text{ state 1NFA}].\end{aligned}$$

Thus (by Theorem 5.1), $8^{f(n)} + 2 \geq 2^{(n-1)/2}$. The result follows.

The proof of (b) uses a **descent argument**. By iteratively applying the assumption, we have, for all n and all k (varying independently), [n state 2DFA] $\subseteq$ L.P.HOM[$f^k(n)$ state 2DFA]. (Here f^k is the composition of k copies of f.) Suppose now, by contradiction, that $f(n) \leq n - 1$ for all $n \geq N_0$ (for some fixed $N_0 \in \mathbb{N}$); then we have:

Claim 1. *For every* $n \geq N_0$, *there exists* $K_n \leq n - N_0$ *such that* $f^{K_n}(n) \leq N_0 - 1$.

Proof. If $n \geq N_0$, then $f(n) \leq n$. As long as $f^k(n) \geq N_0$, we keep applying f, and at every step the number strictly decreases: $f^k(n) > f^{k+1}(n)$ □

Claim 2. *For all* $n \geq N_0$, [n state 2DFA] $\subseteq$ L.P.HOM[$N_0 - 1$ state 2DFA] (*where* N_0 *is the above fixed number*).

Proof. By what we saw in the beginning of the proof of this theorem,

[n state 2DFA] $\subseteq$ L.P.HOM[$f^{K_n}(n)$ state 2DFA]

for all n; and, by Claim 1, $f^{K_n}(n) \leq N_0 - 1$. So Claim 2 follows. □

However, Claim 2 implies that *all* regular languages (when $n \to \infty$) can be recognized by finite automata with a bounded number of states, which is false. □

6. Conclusion

In this paper many upper bounds (inclusions, compressibility) and lower bounds (incompressibility) are given; more results appear in [B6] and [B7]. However, many problems about the state-complexity of finite devices remain open, for example the Sakoda–Sipser conjecture (about the 1NFA $\to$ 2DFA conversion [SS], [Si1]), and other problems stated at the beginning of [B3]. Another long-term goal would be to extend these results to (possibly nonuniform) space-complexity.

Appendix. Definitions of Various Finite-State Devices

In this appendix we give precise definitions of the various finite-state devices used in the paper, their acceptance rules, and related notions; we also prove some fundamental results. This makes the paper self-contained and also handles the problem of the (usually minor) variations in the definitions that appear in the literature.

A1. One-Way Deterministic, Nondeterministic, or Universal Finite Automata

We follow pp. 19–24 and 13–19 of [HU] (except that we also distinguish between partial and complete 1DFAs.)

Definition. *A one-way nondeterministic finite automaton* (**1NFA**) is a structure $\mathbf{N} = (Q, \Sigma, \delta, q_0, F)$, where Q and Σ are finite sets (called "set of states," respectively "input alphabet"), $q_0 \in Q$ (q_0 is called "start state"), and δ is a function from $Q \times \Sigma$ into $P(Q)$, where $P(Q)$ is the power set of Q (δ is called "next-state relation").

A *partial one-way deterministic finite automaton* (**1DFA**) is a 1NFA in which $\delta(q, a)$ contains at most one element of Q, for every $q \in Q$, $a \in \Sigma$. So here δ is a partial function from $Q \times \Sigma$ into Q. A *complete 1DFA* is a partial 1DFA in which $\delta(q, a)$ has exactly one element, for every $q \in Q$, $a \in \Sigma$. So here δ is a total function.

The *state graph* of a 1NFA is the directed labeled graph with vertex set Q, and edges of the form $p \xrightarrow{a} q$ whenever $q \in \delta(p, a)$.

A *computation* of a 1NFA **N** on input $x \in \Sigma^*$ is a walk in the state graph, starting with the vertex q_0, such that when the labels of the edges of the walk are concatenated x is obtained.

A word $x \in \Sigma^*$ is *accepted* by the 1NFA **N** iff there exists ($\exists$) a computation of **N** on input x such that the last state of this computation belongs to F. The language recognized by **N** is the set of all words $\in \Sigma^*$ accepted by **N**.

Definition. A *one-way, universally accepting, nondeterministic finite automaton* (**$\forall$1NFA**) is a structure $\mathbf{A} = (Q, \Sigma, \delta, q_0, F)$ which is exactly like a 1NFA; the notions of *state graph* and of *computation* on an input $x \in \Sigma^*$ are the same as for 1NFAs. However, the definition of acceptance is different: a word $x \in \Sigma^*$ is *accepted* by a $\forall$1NFA **A** iff for every ($\forall$) computation of **A** on input x, the last state of that computation belongs to F. Here the logical subtlety that if there is no computation of **A** on input x (i.e., no walk on **A** starting with q_0 can be labeled by x), then x is accepted, should be observed. The language L recognized by a $\forall$1NFA **A** is the set of words accepted by **A**. Observe that L is recognized by a $\forall$1NFA **A** iff $\Sigma^* - L$ is recognized by the ordinary 1NFA obtained from **A** by replacing F by $Q - F$ (see Theorem 1.1 of [L3]).

Comment. $\forall$1NFAs are just as natural as 1NFAs but in formal language theory they have hardly been used at all. In the control of concurrent processes $\forall$1NFAs would be useful.

A2. One-Way Alternating Finite Automata

We closely follow [BL] and [L1] (where 1AFAs are called "boolean automata"). We need a preliminary definition: A *boolean function* on the set of variables Q is a function from $\{0, 1\}^Q$ into $\{0, 1\}$; here $\{0, 1\}^Q$ is the set of functions from Q to $\{0, 1\}$ (and thus the set of boolean functions on the set of variables Q is denoted by $\{0, 1\}^{\{0,1\}^Q}$). For $q \in Q$, we also denote by q the boolean function whose disjunctive normal form is q; this is the function $(i_1, \ldots, i_n) \in \{0, 1\}^n \to i_q \in \{0, 1\}$, assuming $Q = \{1, \ldots, n\}$.

Definition. A *one-way alternating finite automaton* (**1AFA**) is a structure $\mathbf{A} = (Q, \Sigma, \delta, f_0, F)$, where Q and Σ are finite sets (called "set of states," respectively "input alphabet"), F is a subset of Q (called "set of accept states"), f_0 is a boolean function on the set of variables Q (called "initial function"), and δ is a function

from $Q \times \Sigma$ into $\{0,1\}^{\{0,1\}^Q}$ (called "transition function"). We extend δ to a function δ' from $\{0,1\}^{\{0,1\}^Q} \times \Sigma$ into $\{0,1\}^{\{0,1\}^Q}$ as follows:

$$\delta'(f, a) = f(\delta(q_1, a), \ldots, \delta(q_n, a)),$$

where $Q = \{q_1, \ldots, q_n\}$, $|Q| = n$. We further extend δ' to a function δ^* from $\{0,1\}^{\{0,1\}^Q} \times \Sigma^*$ into $\{0,1\}^{\{0,1\}^Q}$ by induction on the length of the input word: $\delta^*(f, \varepsilon) = f$, and $\delta^*(f, wa) = \delta'(\delta^*(f, w), a)$, for $w \in \Sigma^*$, $a \in \Sigma$, $f \in \{0,1\}^{\{0,1\}^Q}$.

A word $w \in \Sigma^*$ is *accepted* by the 1AFA **A** iff $h(v_F) = 1$, where h is the boolean function $\delta^*(f_0, w)$ and v_F is the characteristic vector of the subset F of Q (i.e., $v_F = (x_1, \ldots, x_n)$ where $x_i = 1$ if $q_i \in F$ and $x_i = 0$ if $q_i \notin F$).

It is useful in the study of 1AFAs to have an analogue of the notion of "computation of **A** on input word w"; this is the notion of computation circuit. The *computation circuit* of **A** on input word $w = a_1 a_2 \cdots a_k$ (with $a_1, \ldots, a_k \in \Sigma$, $k = |w|$) is the following acyclic (combinational) switching circuit: There are $1 + |Q|(|w| + 1)$ gates; we imagine the gates as drawn in columns 1 through $|w| + 1$, with $|Q|$ gates per column; there is an additional output gate at the left end of the circuit; every gate only feeds into gates in the neighboring column to the left (i.e., the information travels from right to left). There is one gate for each $(q, i) \in Q \times \{1, \ldots, |w| + 1\}$, drawn in column i; for $i \leq |w|$ the gate corresponding to (q, i) implements the boolean function $\delta(q, a_i)$. The gate corresponding to $(q, |w| + 1)$ produces the constant value 1 if $q \in F$, and 0 if $q \notin F$. Moreover, we have an additional "output-gate" implementing the initial function f_0. The connections in the circuit are as follows: when $i > 1$, the gate (q, i) feeds into the $|Q|$ gates $\{(p, i-1)/p \in Q\}$ (i.e., the gates in column $i - 1$) along connection edges that point left; moreover, any gate (q, 1) feeds into the output gate (implementing f_0). The following is straightforward to check:

Fact. *The* 1AFA **A** *accepts* w *iff the computation circuit of* **A** *on input* w, *when evaluated, produces a* 1 *at its output gate* f_0.

Finally, let us define important special classes of 1AFAs:

A 1AFA $\mathbf{A} = (Q, \Sigma, \delta, f_0, F)$ is a **Σ_k-1AFA** (for a fixed integer $k \geq 0$) if and only if:

(1) Every state $q \in Q$ is either an "existential state" ($\exists$-state) or a "universal state" ($\forall$-state). By definition, q is an $\exists$-state if, for every $a \in \Sigma$, there exist a set of variables $\{q_{i_1}, \ldots, q_{i_m}\} \subseteq Q$ such that $\delta(q, a) = q_{i_1} \vee \cdots \vee q_{i_m}$; q is a $\forall$-state if, for every $a \in \Sigma$, there exists a set of variables $\{q_{j_1}, \ldots, q_{j_n}\} \subseteq Q$ such that $\delta(q, a) = q_{j_1} \wedge \cdots \wedge q_{j_n}$.

(2) The initial function f_0 has the disjunctive normal form q_0 (for some state $q_0 \in Q$, called "the start state"); moreover, q_0 is a $\exists$-state.

(3) For every word $w \in \Sigma^*$, the computation circuit of **A** on w has the following property: every path (of length $|w|$) in the circuit, starting at any gate $(q, |w| + 1)$ and ending at the gate $(q_0, 1)$, can be factored into $\leq k$ segments such that, within any segment, all the gates are of the same type (i.e., within a segment all gates are $\exists$ or all gates are $\forall$).

We define Π_k-**1AFA**s similarly: they differ from Σ_k-1AFAs only in the fact that q_0 is now a $\forall$-state.

A3. Two-Way Deterministic or Nondeterministic Finite Automata

We follow pp. 36–42 of [HU] (except that we use endmarkers as in [LLS], [SS], [Si1], or as for Turing machines).

Definition. A *two-way nondeterministic finite automaton* (**2NFA**) is a structure $\mathbf{N} = (Q, \Sigma, \delta, q_0, F)$ where Q and Σ are finite sets (called "set of states," respectively "input alphabet"), $q_0 \in Q$ (q_0 is called "start state"), $F \subseteq Q$ (F is called "set of accept states"), and δ is a function from $Q \times (\Sigma \cup \{\langle, \rangle\})$ into $P(Q \times \{-1, +1\})$ (δ is called "next-state relation"); here $P(\cdots)$ denotes the power set. A 2NFA has two additional special tape symbols, namely $\langle$ and $\rangle$ (the left, respectively right, endmarker) which do not belong to Σ.

A *two-way deterministic finite automaton* (**2DFA**) is a 2NFA for which $\delta(q, a)$ contains at most one element, for every $q \in Q$, $a \in \Sigma \cap \{\langle, \rangle\}$.

A *current configuration* of the 2NFA **N** on input $w \in \Sigma^*$ is a string $\langle uqv\rangle$, where $uv = w$, $q \in Q$ (the current state), and the reading head is positioned on the leftmost letter of v (or on $\rangle$ if $v = \varepsilon$). We also allow a configuration $q\langle w\rangle$ (when the head is positioned on the left endmarker), and the configuration $\langle w\rangle q$ (when the reading head has moved off the tape on the right).

Suppose the current configuration of **N** on input w is $\langle uqav_1\rangle$, where a is the leftmost letter of $v \neq \varepsilon$ (i.e., let $v = av_1$), and suppose $(p, e) \in \delta(q, a)$, $e \in \{-1, +1\}$. If $e = +1$, then a *next configuration* is $\langle uapv_1\rangle$. If $e = -1$, let b be the rightmost letter of u (and write $u = u_1 b$), or let $b = \langle$ if $u = \varepsilon$; then a next configuration is $\langle u_1 pbv\rangle$. If the current configuration is $q\langle w\rangle$, and $(p, +1) \in \delta(q, \langle)$ then a next configuration is $\langle pw\rangle$. If c_1 is a configuration of **N** on input w, and c_2 is a next configuration, we write $c_1 \vdash c_2$; we also use the reflexive-transitive closure $\vdash^*$ of $\vdash$.

An input $w \in \Sigma^*$ of a 2NFA **N** is accepted iff $q_0\langle w\rangle \vdash^* \langle w\rangle f$, for some $f \in F$. (Note that it is w that is accepted, not $\langle w\rangle$, since the endmarkers are not considered as inputs but are part of the machine.)

It is convenient, for notational purposes, also to consider "fragmentary" configurations of the form xqy, where $q \in Q$, $x \in \Sigma^* \cup \langle\Sigma^*$, and $y \in \Sigma^* \cup \Sigma^*\rangle$; in other words, here we allow one or both of the two endmarkers to be absent. The relations $\vdash$ and $\vdash^*$ can be applied to fragmentary configurations as well (the definition is the same as for the usual configurations).

The *computation graph* of a 2NFA **N** on input $w \in \Sigma^*$ is the following directed labeled graph:

the set of *vertices* is $Q \times \{0, 1, \ldots, |w| + 2\}$;

the edges are of the form $(q, i) \xrightarrow{a} (p, i + 1)$ whenever $i \leq |w| + 1$, $(p, +1) \in \delta(q, a)$, and of the form $(q, i) \xrightarrow{a} (p, i - 1)$ whenever $i \geq 1$, $(p, -1) \in \delta(q, a)$; here a is the ith letter of w (or $a = \langle$ if $i = 0$, or $a = \rangle$ if $i = |w| + 1$).

A *computation* of the 2NFA **N** on input $w \in \Sigma^*$ is a walk in the computation graph of **N** on w, starting with the vertex $(q_0, 0)$. The computation is *accepting* if it ends in a vertex $(f, |w| + 2)$, for some $f \in F$.

The following simple results about 2NFAs are used in this paper:

Fact A3.0. *Every* 2NFA *with* n *states is equivalent to a* 2NFA *with* n *states which has just one accept state. Moreover, if the former* 2NFA *is deterministic, then so is the latter.*

Proof. When a 2NFA **N** starting in configuration $q_0\langle w\rangle$ can reach a configuration $\langle w\rangle f$ with $f \in F$, then no further transitions are possible. Let f_1 be any element of F. We now change **N** by replacing all elements of the form $(f, +1)$ in $\delta(q, \rangle)$ (where $q \in Q$, $f \in F$), by $(f_1, +1)$ (i.e., any transition $\langle wq\rangle \vdash \langle w\rangle f$ is replaced by $\langle wq\rangle \vdash \langle w\rangle f_1$, for the fixed element f_1 of F). Now, repalce F by $\{f_1\}$; the total set of states Q remains unchanged. If **N** was deterministic, then the new automaton will also be deterministic. It is straightforward to check that $w \in \Sigma^*$ is accepted by the new automaton iff w was accepted by **N**. □

Fact A3.1. *If* $L \subseteq \Sigma^*$ *is recognized by a* 2NFA *with* n *states, then* L^{rev} *is also recognized by a* 2NFA *with* n *states.*

Proof. We replace $\mathbf{N} = (Q, \Sigma, \delta, q_0, \{f\})$ (with single accept state, by Fact A3.0), by $\mathbf{N}' = (Q, \Sigma, \delta', f, \{q_0\})$, with δ' defined by $(p, e) \in \delta'(q, a)$ iff $(q, e) \in \delta(p, a)$. (This is the same idea as for reversing a 1NFA, but we do not need a new start state, thanks to Fact A3.0.) □

Fact A3.2. *If* $L \subseteq \Sigma^*$ *is recognized by a* 2DFA *with* n *states, then* L^{rev} *is recognized by a* 2DFA *with* $\leq n + 2$ *states.*

Proof. Let $\mathbf{A}_1 = (Q, \Sigma, \delta, q_0, F)$ be a 2DFA, with endmarkers $\langle$ and $\rangle$, recognizing L; by Fact A3.0 we may assume $F = \{f_1\}$. A 2DFA for L^{rev} is $\mathbf{A}_2$, obtained from $\mathbf{A}_1$ by adding two new states i and f_2; the idea is that $\mathbf{A}_2$ first sweeps over $\langle w\rangle$ (where $w \in \Sigma^*$ is the input) from left to right, in state i; then $\mathbf{A}_2$ simulates $\mathbf{A}_1$ with "left" and "right" (and also $\langle$ and $\rangle$) interchanged; finally (at the left end of w) when $\mathbf{A}_1$ accepts, $\mathbf{A}_2$ goes to state f_2 and again sweeps over $\langle w\rangle$ from left to right. In more detail: $\mathbf{A}_2 = (Q \cup \{i, f_2\}, \Sigma, \delta_2, i, \{f_2\})$ where i and f_2 do not belong to Q, and where δ_2 is defined by $\delta_2(i, a) = (i, +1)$ for all $a \in \Sigma \cup \{\langle\}$, and $\delta_2(i, \rangle) = (p_1, -1)$ where $p_1 \in Q$ is such that $(p_1, +1) = \delta_1(q_0, \langle)$; next, for all $q \in Q$, $a \in \Sigma$ we define $\delta_2(q, a) = (p, -d)$ whenever $\delta_1(q, a) = (p, d)$; at the right end we have, for all $q \in Q$, $\delta_2(q, \rangle) = (p, -d)$ whenever $\delta_1(q, \langle) = (p, d)$; at the left end we have, for all $q \in Q$, $\delta_2(q, \langle) = (p, -d)$ whenever $\delta_1(q, \rangle) = (p, d)$ with $p \neq f_1$; finally, $\delta_2(q, \langle) = (f_2, +1)$ if $\delta_1(q, \rangle) = (f_1, +1)$; and $\delta_2(f_2, a) = (f_2, +1)$ for all $a \in \Sigma \cup \{\langle, \rangle\}$. □

Definition. The *global state transitions* of a 2NFA **A** on a word $u \in (\Sigma \cup \{\langle, \rangle\})^+$ are the following four relations on Q (see [B1], where however a slightly different kind of 2NFA was used):

$[\rightarrow u \rightarrow] \subseteq Q \times Q$ is the relation defined by $(q_1, q_2) \in [\rightarrow u \rightarrow]$ iff $q_1 u \vdash^* u q_2$.

$[\rightleftarrows u] \subseteq Q \times Q$ is the relation defined by $(q_1, q_2) \in [\rightleftarrows u]$ iff $(\exists q \in Q)(q_1 u \vdash^* qu$ and $(\exists \sigma \in \Sigma \cup \{\langle, \rangle\})(\sigma q u \vdash q_2 \sigma u))$ (i.e., there exists a computation of **A** starting at the left end of u in state q_1; during this computation, the reading head of **A** stays on u, and eventually leaves u on the left end, in state q_2).

$[u \leftrightarrows] \subseteq Q \times Q$ is the relation defined by $(q_1, q_2) \in [u \leftrightarrows]$ iff $vq_1 a \vdash^* vaq_2$, where $u = va$, $a \in \Sigma \cup \{\langle, \rangle\}$.

$[\leftarrow u \leftarrow] \subseteq Q \times Q$ is the relation defined by $(q_1, q_2) \in [\leftarrow u \leftarrow]$ iff $(\exists q \in Q) \times (vq_1 a \vdash^* qva$ and $(\exists \sigma \in \Sigma \cup \{\langle, \rangle\})(\sigma q v a \vdash q_2 \sigma q a))$, where $va = u$, $a \in \Sigma \cup \{\langle, \rangle\}$.

We have the following fact (see [B1]).

Fact A3.3. *If* $u, v \in (\Sigma \cup \{\langle, \rangle\})^+$, *then we have, for the concatenation* uv:

$$[\rightarrow uv \rightarrow] = [\rightarrow u \rightarrow]([\rightleftarrows v][u \leftrightarrows])^*[\rightarrow v \rightarrow].$$

$$[\rightleftarrows uv] = [\rightleftarrows u] \cup [\rightarrow u \rightarrow]([\rightleftarrows v][u \leftrightarrows])^*[\rightleftarrows v][\leftarrow u \leftarrow].$$

$$[uv \leftrightarrows] = [v \leftrightarrows] \cup [\leftarrow v \leftarrow]([u \rightleftarrows][\leftrightarrows v])^*[u \leftrightarrows][\rightarrow v \rightarrow].$$

$$[\leftarrow uv \leftarrow] = [\leftarrow v \leftarrow]([u \leftrightarrows][\rightleftarrows v])^*[\leftarrow u \leftarrow].$$

Notation: *Juxtaposition of relations denotes composition of relations (defined in the usual way). The star *, applied to a relation, denotes reflexive-transitive closure.*

Below, if $R \subseteq Q \times Q$ is a relation and $q \in Q$, then $(q)R = \{s \in Q/(q, s) \in R\}$.

We now prove the following improvement of Shepherdson's theorem (of [Sh]):

Theorem A3.4. *If* $L \subseteq \Sigma^*$ *is recognized by a* 2NFA (*or a* 2DFA) *with* n *states, then* L *is recognized by a* partial 1DFA *with* $\leq 2^{(n-1)^2+n} - 2^{(n-1)^2}$ *states (resp.* $\leq n^n$ *states).*

Proof. From a 2NFA $\mathbf{A}_2 = (Q, \Sigma, \delta_2, q_0, F)$ we construct the following partial 1DFA $\mathbf{A}_1$, which recognizes the same language L:

State set of $\mathbf{A}_1$:

$\{(P_x, f_x)/x \in \Sigma^*, P_x = (q_0)[\rightarrow \langle x \rightarrow] \subseteq Q, P_x \neq \varnothing,$ and $f_x = [\langle x \leftrightarrows] \cap (Q - P_x) \times (Q - P_x)\}$

(this is almost the same as Shepherdson's construction in [Sh]; the only difference is that here f_x is restricted to states in $Q - P_x$).

Start state: $(P_\varepsilon, f_\varepsilon)$.

Set of accept states: $\{(P_x, f_x)/x \in \Sigma^*, (q_0)[\rightarrow \langle x \rangle \rightarrow] \cap F \neq \varnothing\}$.

Next-state function: for a current state (P_x, f_x) and letter $a \in \Sigma$, the next state is (P_{xa}, f_{xa}) if $P_{xa} \neq \varnothing$ (there is no next state if $P_{xa} = \varnothing$).

We show in the next two lemmas that the next-state function (which is a partial function) is well defined, i.e., that the knowledge of (P_x, f_x) (here x is not explicitly known) and a determines at most one state (P_{xa}, f_{xa}).

Lemma. *Given* (P_x, f_x) *and* $a \in \Sigma \cup \{\rangle\}$, *the set* $(q_0)[\rightarrow\langle xa\rightarrow]$ *is uniquely determined. In particular,* P_{xa} *is unique (for* $a \in \Sigma$*) and* $[\rightarrow\langle x\rangle\rightarrow]$ *is uniquely determined (for* $a = \rangle$*).*

Proof. We show that, for every $q \in Q$, we can uniquely determine whether or not $q \in (q_0)[\rightarrow\langle xa\rightarrow]$. By definition, $q \in (q_0)[\rightarrow\langle xa\rightarrow]$ iff $q_0\langle xa \vdash^* \langle xaq$. During any such computation of $\mathbf{A}_2$, the first time the reading head gets to the position of a at the right end of $\langle xa$, the state is in P_x. For any such computation (starting in configuration $q_0\langle xa$ and ending in configuration $\langle xaq$), consider the *last* time that the reading head visits the position of a in a state in P_x; let $q_1 \in P_x$ be that state. Now, our computation $q_0\langle xa \vdash^* \langle xaq$ can be broken into two parts:

(1) $q_0\langle xa \vdash^* \langle xq_1a$ and
(2) $\langle xq_1a \vdash^* \langle xaq$.

Since in (2), no state in P_x occurs at the position of a, we have

$$(q_1, q) \in ([\rightleftarrows a]([\langle x\leftrightarrows] \cap (Q - P_x) \times (Q - P_x)))^*[\rightarrow a\rightarrow]$$
$$= ([\rightleftarrows a]f_x)^*[\rightarrow a\rightarrow].$$

Thus we have $q \in (q_0)[\rightarrow\langle xa\rightarrow]$ iff there exists

$$q_1 \in P_x \text{ such that } (q_1, q) \in ([\rightleftarrows a]f_x)^*[\rightarrow a\rightarrow].$$

Obviously, this condition can be checked if P_x, f_x, and a are known. □

Lemma. *Given* (P_x, f_x) *and* $a \in \Sigma \cup \{\rangle\}$, *the relation* $f_{xa} \subseteq (Q - P_{xa}) \times (Q - P_{xa})$ *is uniquely determined.*

Proof. We show that, for every $p, q \notin P_{xa}$, we can uniquely determine whether $(p, q) \in f_{xa}$. By the previous lemma, P_{xa} is uniquely determined. By definition, $(p, q) \in f_{xa}$ iff $p, q \notin P_{xa}$ and $\langle xpa \vdash^* \langle xaq$. Observe that such a computation does not use any state in P_x at the position of a in $\langle xa$ (otherwise q would belong to P_{xa}). It follows that $(p, q) \in f_{xa}$ iff $(p, q) \in [a\leftrightarrows] \cup [\leftarrow a\leftarrow](f_x[\rightleftarrows a])^*f_x[\rightarrow a\rightarrow]$; i.e., when $[\langle x\leftrightarrows]$ is used, we never need apply it to a state in P_x. Therefore $(p, q) \in f_{xa}$ iff $p, q \notin P_{xa}$ and $(p, q) \in [a\leftrightarrows] \cup [\leftarrow a\leftarrow](f_x[\rightleftarrows a])^*f_x[\rightarrow a\rightarrow]$; this can be checked if a, f_x, and P_{xa} (and P_{xa} is determined by a, f_x, P_x) are known. □

From the lemmas it follows that $\mathbf{A}_1$ is indeed a partial 1DFA. It can easily be checked that $w \in \Sigma^*$ is accepted by $\mathbf{A}_1$ iff $(q_0)[\rightarrow\langle x\rangle\rightarrow] \cap F = P_{x\rangle} \cap F \neq \varnothing$; this is iff w is accepted by the 2NFA $\mathbf{A}_2$. Finally, from the construction we can see that the number of states of $\mathbf{A}_1$ is

$$\leq \sum_{k=1}^{n} \binom{n}{k} 2^{(n-k)^2}.$$

This is obviously $\leq (2^n - 1)2^{(n-1)^2}$. A more refined upper bound is $n\binom{n}{n/2}2^{(n-1)^2}$.

Here the number of states of $\mathbf{A}_2$ is $|Q| = n$.

If $\mathbf{A}_2$ is a 2DFA, then every f_x is a partial function on $Q - \{(q_0)[\rightarrow\langle x\rightarrow]\}$, which is a set of $n - 1$ elements. In that case the number of states of the partial 1DFA $\mathbf{A}_1$ is $\leq nn^{n-1} = n^n$. □

A4. Two-Way Alternating Finite Automata

A *two-way alternating finite automaton* (**2AFA**) is a structure $\mathbf{A} = (Q, \Sigma, \delta, f_0, F)$ where Q and Σ are finite sets (called "set of states," respectively "input alphabet"), F is a subset of Q (called "set of accept states"), f_0 is a boolean function over the set of variables $Q \times \{+1\}$ (called "initial function"), and δ is a function from $Q \times (\Sigma \cup \{\langle, \rangle\})$ into the set of boolean functions over the set of variables $Q \times \{-1, +1\}$; here, as for 2NFAs, $\langle$ and $\rangle$ are the endmarkers of the tape (at the left (resp. the right) end).

Just like for 1AFAs, we define the notion of a *computation circuit* $C_{\langle w\rangle}$ of the 2AFA **A** on a tape $\langle w\rangle \in \langle\Sigma^*\rangle$ (where $w \in \Sigma^*$ is the input): $C_{\langle w\rangle}$ is an asynchronous sequential circuit (not necessarily combinational this time) with $1 + |Q|(|w| + 2)$ gates. There is one gate (that we picture at the left end of the circuit) implementing the boolean function f_0; this gate has a single output port (on its left side) and $|Q|$ input ports. The other gates have labels of the form $(q, i) \in Q \times \{0, 1, \ldots, |w| + 1\}$, and are pictured in columns $0, 1, \ldots, |w|, |w| + 1$, with $|Q|$ gates per column. A gate (q, i) implements the boolean function $\delta(q, a_i)$: $\{0, 1\}^{Q \times \{-1, +1\}} \rightarrow \{0, 1\}$ over the set of boolean variables $Q \times \{-1, +1\}$; here a_i is the ith letter of the input string w (and for $i = 0$, $a_i = \langle$, and for $i = |w| + 1$, $a_i = \rangle$). The gate (q, i) has $2 \cdot |Q|$ input ports labeled by the set of boolean variables $Q \times \{-1, +1\}$. The input port $(p, -1)$ (with $p \in Q$) of gate (q, i) is connected to the output port of gate $(p, i - 1)$; note that a left-move leads from position i to position $i - 1$, and therefore the input port $(p, -1)$ on (q, i) is used. Similarly, the input port $(p, +1)$ of gate (q, i) is connected to the gate $(p, i + 1)$. The gate (q, i) has a single output port (since $\delta(q, a_i)$ is a function into the boolean values $\{0, 1\}$); this port is connected to the input ports labeled $(q, +1)$ of all the gates $(p, i - 1)$ (as p ranges over Q), and to the input ports $(q, -1)$ of all the gates $(p, i + 1)$ (as p ranges over Q).

Remark. The gates in column 0 (where $a_0 = \langle$) and in column $|w| + 1$ (where $a_{|w|+1} = \rangle$) are a little special: In column 0, all input ports $(q, -1)$ (corresponding to a left-move on the left endmarker $\langle$) are set to the boolean value 0; the output ports in column 0 feed into the gate implementing f_0. In column $|w| + 1$ the input ports $(q, +1)$ are set to the boolean value 1 when $q \in F$, and to 0 when $q \notin F$.

We also use the computation circuit $C_{\langle w}$ of the 2AFA **A** on a tape $\langle w \in \langle\Sigma^*$ (without a right endmarker). It differs from the circuit $C_{\langle w\rangle}$ for $\langle w\rangle$ only by the fact that column $|w| + 1$ is absent. $C_{\langle w}$ has $|Q|$ input ports (namely the right input

ports of column $|w|$, corresponding to the set of boolean variables $Q \times \{+1\}$), and it has $|Q| + 1$ output ports (namely one port on the left, at the f_0 gate, and the $|Q|$ output ports at the right of column $|w|$). (Note that the circuit $C_{\langle w \rangle}$ has no input ports, and it has just one output port, namely the f_0 gate on the left.) Thus the circuit $C_{\langle w}$ (for $w \in \Sigma^*$) implements a boolean function $B_{\langle w}$: $\{0,1\}^{|Q|} \to \{0,1\}^{|Q|+1}$ (with $|Q|$ boolean input variables and $|Q| + 1$ boolean output variables).

The input $w \in \Sigma^*$ is *accepted* by **A** iff the computation circuit $C_{\langle w \rangle}$ produces a stable boolean value 1 at the output of the special leftmost gate (implementing f_0). See, e.g., [BS] for analysis of the stable values of an asynchronous sequential circuit.

We now show the following extension of [LLS] for our more general 2AFAs.

Theorem A4.1. *If* $L \subseteq \Sigma^*$ *is recognized by a* 2AFA *with* n *states, then* L *is recognized by a* complete 1DFA *with* $\leq 2^{(n+1)2^n}$ *states.*

Proof. From a 2AFA $\mathbf{A}_2 = (Q, \Sigma, \delta_2, f_0, F)$ we construct a complete 1DFA $\mathbf{A}_1$, described as follows:

Set of states of $\mathbf{A}_1$: $\{B_{\langle w}/w \in \Sigma^*\}$ (where $B_{\langle w}$ is the boolean function defined above).
Start state: $B_{\langle}$.
Set of accept states: $\{B_{\langle w}/$the gate f_0 of $C_{\langle w \rangle}$ outputs the boolean value $1\}$.
Next-state function: for $a \in \Sigma$ and a current state $B_{\langle w}$, the next state is $B_{\langle wa}$.

To see that the next-state function is well defined, we view $B_{\langle w}$ and a as black boxes, as in Figure 2. The black box corresponding to a is made in the same way as a column (corresponding to the input letter a) in the computation circuits $C_{\langle w}$ or $C_{\langle w \rangle}$.

From the definition of the accept states of $\mathbf{A}_1$, it can be immediately seen that $\mathbf{A}_1$ recognizes the same language as $\mathbf{A}_2$. Also, clearly $\mathbf{A}_1$ has $\leq (2^{n+1})^{2^n} = 2^{(n+1)2^n}$ states, where $n = |Q|$. □

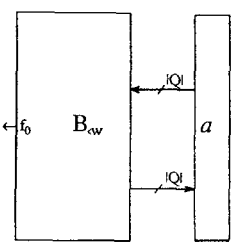

Fig. 2. Black boxes for $B_{\langle w}$ and a.

References

[AU] A. Aho, J. Ullman, *The Theory of Parsing, Translation and Compiling*, Vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[B1] J. C. Birget, Concatenation of inputs in a two-way automaton, *Theoret. Comput. Sci.*, **63** (1989), 141–156.

[B2] J. C. Birget, Positional simulation of two-way automata: proof of a conjecture of R. Kannan, and generalizations, *J. Comput. System Sci.* (special issue on *STOC 89*), **45** (1992), 154–179.

[B3] J. C. Birget, Two-way automata and length-preserving homomorphisms, Report #109, Dept. of Computer Science, University of Nebraska (1990) (submitted).

[B4] J. C. Birget, The minimum automaton of certain languages (in preparation).

[B5] J. C. Birget, Strict local testability of the finite control of two-way automata and of regular picture description languages, *Internat. J. Algebra Comput.*, **1** (1991), 161–175.

[B6] J. C. Birget, Partial order on words, minimal elements of a regular language, and state-complexity, *Theoret. Comput. Sci.* (to appear).

[B7] J. C. Birget, Intersection and union of regular languages and state-complexity, *Inform. Process. Lett.*, **43** (1992), 185–190.

[BH] M. Blum, C. Hewitt, Automata on a 2-dimensional tape, *Proc. 8th IEEE Symp. on Switching and Automata Theory*, 1965, pp. 155–160.

[BG] R. Book, S. Greibach, Quasi-realtime languages, *Math. System Theory*, **4** (1970), 97–111.

[BL] J. Brzozowski, E. Leiss, On equations for regular languages, finite automata, and sequential networks, *Theoret. Computer Sci.*, **10** (1980), 19–35.

[BS] J. Brzozowski, C. Seger, Advances in asynchronous circuit theory, Part 1, *Bull. EATCS*, **42** (1990), 198–249.

[CKS] A. Chandra, D. Kozen, L. Stockmeyer, Alternation, *J. Assoc. Comput. Mach.*, **28** (1981), 114–133.

[CS] A. Chandra, L. Stockmeyer, Alternation, *Proc. 17th IEEE Symp. on Foundations of Computer Sci.*, 1976, pp. 98–108.

[C] M. Chrobak, Finite automata and unary languages, *Theoret. Comput. Sci.*, **47** (1986), 149–158.

[E] S. Eilenberg, *Automata, Languages, and Machines*, Vol. A, Academic Press, New York, 1974.

[H] F. C. Hennie, One-tape off-line Turning machine computations, *Inform. and Control*, **8** (1965), 553–578.

[HU] J. Hopcroft, J. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesely, Reading, MA, 1979.

[Ka] R. Kannan, Alternation and the power of non-determinism, *Proc. 15th ACM Symp. on Theory of Computing*, 1983, 344–346.

[Ko] D. Kozen, On parallelism in Turing machines, *Proc. 17th IEEE Symp. on Foundations of Computer Sci* 1976, pp. 89–97.

[LLS] R. Ladner, R. Lipton, L. Stockmeyer, Alternating pushdown automata, *Proc. 19th IEEE Symp. on Foundations of Computer Sciences* 1978, pp. 92–106, and *SIAM J. Comput.*, **13**(1) (1984), 135–155.

[L1] E. Leiss, Succinct representation of regular languages by boolean automata, *Theoret. Comput. Sci.*, **13**(1981), 323–330.

[L2] E. Leiss, Succinct representation of regular languages by boolean automata, II, *Theoret. Comput. Sci.*, **38** (1985), 133–136.

[L3] E. Leiss, A class of tractable unrestricted regular expressions, *Theoret. Comput. Sci.*, **35** (1985), 313–327.

[MP] R. McNaughton, S. Papert, *Counter-free Automata*, MIT Press, Cambridge, MA, 1971.

[MF] A. R. Meyer, M. J. Fischer, Economy of description by automata, grammars, and formal systems, *Proc. 21st IEEE Symp. on Switching and Automata Theory*, 1971, pp. 188–191.

[RS] M. Rabin, D. Scott, Finite automata and their decision problems, *IBM J. Res. Develop.*, **3** (1959), 114–125; also in E. F. Moore (ed.), *Sequential Machines: Selected Papers*, Addison-Wesley, Reading, MA, 1964.

[SS] W. Sakoda, M. Sipser, Non-determinism and the size of two-way automata, *Proc. 10th ACM Symp. on Theory of Computing*, 1978, pp. 275–286.

[Sh] J.C. Shepherdson, The reduction of two-way automata to one-way automata, *IBM J. Res. Develop.*, **3** (1959), 198–200; also in E. F. Moore (ed.), *Sequential Machines: Selected Papers*, Addison-Wesley, Reading, MA, 1964.

[Si1] M. Sipser, Lower bounds on the size of sweeping machines, *Proc. 11th ACM Symp. on Theory of Computing*, 1979, pp. 360–364; and *J. Comput. System Sci.*, **21** (1980), 195–202.

[Si2] M. Sipser, Halting space-bounded computations, *Theoret. Comput. Sci.*, **10** (1980), 335–338 also *Proc. 19th IEEE Symp. on Foundations of Computer Science*, 1978).

[V] M. Vardi, A note on the reduction of two-way automata to one-way automata, *Inform. Process. Lett.*, **30** (1989), 261–264.

Received December 24, 1990, and in revised form August 15, 1991, and September 29, 1991, and in final form March 23, 1992.