NONDETERMINISM AND THE SIZE
OF TWO WAY FINITE AUTOMATA

by

William J. Sakoda and Michael Sipser

Memorandum No. UCB/ERL M78/34

May 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720
§1.1 INTRODUCTION

An important goal of the theory of computation is the classification of languages according to computational difficulty. Classes such as P, NP, and LOGSPACE provide a natural framework for this, though it is a fundamental open problem to demonstrate languages distinguishing them. The complete languages of Cook, Karp, and others [1-7] are candidates for such languages in the sense that, if the classes are in fact different, these languages witness the difference.

We consider two questions on regular languages resembling these open problems. One of these questions concerns 2-way non-deterministic (2n) and 2-way deterministic (2d) finite automata:

For every 2nfa M, is there an equivalent 2dfa with only polynomially more states than M?

Let \( s_{2n-2d}(n) \) be the least integer such that for every n-state 2nfa there is an equivalent \( s_{2n-2d}(n) \) state 2dfa. The question above can then be restated: Is \( s_{2n-2d}(n) \) bounded above by a polynomial \( p(n) \)?

Here is a summary of our results. In section 2.2 we present a sequence of languages \(<C_1, C_2, \ldots>\) which is complete in the sense that in order to settle the question of polynomial boundedness of \( s_{2n-2d} \), it suffices to determine the size of 2dfa required by each language \( C_n \): \( s_{2n-2d}(n) \) is bounded above by a polynomial \( p(n) \) iff the size of 2dfa required by \( C_n \) is bounded above by some (other) polynomial \( p'(n) \).

In section 2.3 we present languages \(<B_1, B_2, \ldots>\) which are complete with respect to 1-way non-deterministic (1n) to 2-way deterministic conversion: \( B_n \) is an n-state 1n language, and requires the largest 2dfa of any n-state 1n language. We conjecture that the size of 2d acceptors for \( B_n \) is not bounded above by any polynomial \( p(n) \). This conjecture is the starting point for the investigation described in section 4.

In section 4, we consider certain restricted forms of two way automata. Techniques are developed which yield tight lower bounds on the size of such acceptors of the complete languages.

Section 3 presents a convenient notation for comparing succinctness of description of different models of automata. This notation, together with a reducibility, is used to state an analogy between the problems studied in this paper and the \( P = \mathit{NP} \) question.

§1.2 A MAP

2: Completeness Results.

2.1: Definitions of the languages \( C_n, B_n \), and remarks about our models of finite automata.

2.2: Completeness of \(<C_1, C_2, \ldots>\) for \( 2n-2d \).

2.3: Completeness of \(<B_1, B_2, \ldots>\) for \( 1n-2d \).

3: An analogy to \( P = \mathit{NP} \).

4: Lower Bounds.

4.1: One way automata.

4.2: Restricted two way automata (parallel machines).

4.3: Restricted two way automata (series machines).

5: Related Work

Presented at the Tenth Annual Symposium on Theory of Computing, May 1978.
§ 2.1 DEFINITIONS OF LANGUAGES \( C \) and \( B \)

In this section we present the languages \( <C_1,C_2,\ldots> \) which are complete with respect to \( 2n-2d \) conversion, and the languages \( <B_1,B_2,\ldots> \) which are complete for \( 1n-2d \) conversion. Statements and proofs of the completeness of \( C \) and \( B \) follow in sections 2.2 and 2.3. A comment about the models of finite automata we use appears at the end of this section.

**Definition of language \( C \):**

(1) Let the alphabet \( \Gamma \) be the graphs consisting of \( n \) left nodes and \( n \) right nodes. Directed arcs may join any distinct pair of nodes. Figure 1 shows three members of \( \Gamma \).

\[ g_1 \quad g_2 \quad g_3 \]

Figure 1

Given a sequence of graphs \( g_1g_2\ldots g_k \in \Gamma \), the catenated graph \( g_1g_2\ldots g_k \) is obtained by identifying adjacent left and right nodes in the sequence. For example, the graphs in Figure 1 catenate to yield the solid arcs in Figure 2.

\[ \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \]

Figure 2

(2) The language \( C \) consists of all sequences of graphs \( g_1|g_2|\ldots|g_k \in \Gamma \) such that \( g_1|g_2|\ldots|g_k \) has a directed path from a node in the first column to a node in the last column. In Figure 2, the dotted path witnesses the fact that \( g_1|g_2|g_3 \) is in \( C \).

**Definition (of language \( B \)):**

(1) The alphabet \( \Sigma \) is a subset of \( \Gamma \). A graph \( g \) from \( \Sigma \) is in \( \Sigma \) if every arc in \( g \) is directed left to right.

(2) The string \( g_1|g_2|\ldots|g_k \) is in \( B \) if the catenated graph \( g_1|g_2|\ldots|g_k \) has a path from the leftmost column to the rightmost column.

**Definition:** A 2-way non-deterministic finite automaton \( M \) is defined by a 7-tuple \( \langle Q, \Sigma, I, S, B, Q_0, F \rangle \). An input string \( a_1a_2\ldots a_k \) from \( \Sigma \) is presented delimited by left and right endmarkers \( \llbracket \) and \( \rrbracket \). The automaton is started on the symbol \( \llbracket \) in one of the states from the initial state set \( Q_0 \). The transition function \( \delta: (\Sigma \cup \{\llbracket, \rrbracket\}) \rightarrow \mathcal{P}(Q \times \{L,R\}) \) (\( \mathcal{P} \) denoting power set) defines the moves of \( M \); for example if \( a\llbracket L \rrbracket \) then \( M \), scanning input symbol \( a \) in state \( q \) may move left (right, if \( R \) appears instead of \( L \) ) and transfer to state \( q' \). \( M \) accepts if from some initial configuration there is a sequence of moves which causes it to move onto the right endmarker into a final state \( \llbracket \rrbracket \). A 2-way deterministic finite automaton \( (2dfa) \) is a 2nfa where the transition function is never multiply defined; that is \( \delta: (\Sigma \cup \{\llbracket, \rrbracket\}) \rightarrow \mathcal{P}(Q \times \{L,R\}) \) or \( \phi \).

**Definition:** A 1-way non-deterministic finite automaton \( (1nfa) \) is defined by a 5-tuple \( \langle Q, \Sigma, B, Q_0, F \rangle \). \( Q_0 \) is the set of initial states. \( \delta: Q \times \Sigma \rightarrow \mathcal{P}(Q) \) is the transition function.

**Definition:** A 1-way deterministic finite automaton \( (1dfa) \) is defined by a 5-tuple \( \langle Q, I, \delta, q_0, F \rangle \). \( q_0 \) is the initial state. \( \delta: Q \times I \rightarrow Q \) is the (everywhere defined) transition function.

§ 2.2 2N-2D: \( <C_1,C_2,\ldots> \) is COMPLETE

**Theorem 2.2:** The size of 2d acceptors for the languages \( C \) grows polynomially iff state expansion for \( 2n-2d \) conversion is polynomial. More specifically: Let \( c(n) \) denote the size of 2dfa required by \( C \).

(i): \( c(n) \leq s_{2n-2d}(2n) \).

(ii): \( s_{2n-2d}(n) \leq c(2n) \).

**Proof of (i):** \( C \) is accepted by a \( 2n \)-state 2nfa. \( \square \)

**Proof of (ii), Overview:** An \( n \)-state non-deterministic machine \( N \) is given. We can assume that there is a \( c(2n) \)-state 2dfa \( G \) for the language \( C_{2n} \). We will show that there is a \( c(2n) \)-state 2dfa \( D \) which is equivalent to \( N \).

There are 2 parts to this proof. The first part shows a way of encoding computations of the machine \( N \) as \( \Gamma_{2n} \)-graphs (i.e., graphs from \( \Gamma_{2n} \)). Figure 3 illustrates this encoding.

To each string \( x = \llbracket a_1a_2\ldots a_k \rrbracket \) we associate a catenated sequence of \( \Gamma_{2n} \)-graphs \( g(x) = g(\llbracket \rrbracket) | g(a_1) | g(a_2) | \ldots | g(a_k) | g(\rrbracket) \) in such a way that \( x \) is accepted by \( N \) iff \( g(x) \in C_{2n} \). A possible stra-
strategy for recognizing $L(N)$ is then: "On input $x$, test the catenated sequence of graphs $g(x)$ for membership in $C_{2n}$; accept if and only if $g(x) \in C_{2n}$." The second part of the proof shows how to obtain the required 2dfa $D$ by using this strategy.

Proof of (ii), Part 1: An $n$-state 2nfa $N$ is given. We will give a catenated sequence of $\Gamma_{2n}$ — graphs

$g = g(\text{left}) \cdot g(a_1) \cdot g(a_2) \cdot \ldots \cdot g(a_k) \cdot g(\text{right})$ which encodes the computation of $N$ on input $\text{left} a_1 a_2 \ldots a_k \text{right}$. The encoding we wish to establish is:

**Lemma:** There is a path from the leftmost to the rightmost column in $g$ (that is, $g \in C_{2n}$) iff $N$ accepts input $\text{left} a_1 a_2 \ldots a_k \text{right}$.

We now define $g$ to satisfy the Lemma. Let $q_1, q_2, \ldots, q_n$ enumerate the states of $N$.

**Definition of $g(b)$:** With each symbol $b \in \Sigma \cup \{\text{left}, \text{right}\}$, associate a graph $g(b) \in \Gamma_{2n}$. Let the nodes of $g(b)$ be named as in Figure 4.
This concludes the definition of \( g \).

We now show that \( g \) satisfies the Lemma. Moves made by \( N \) correspond to paths through the catenated graph
\[
\begin{align*}
&g(\text{-}) | g(a_1) | \cdots | g(a_k) | g(\text{-})
\end{align*}
\]
and conversely. Figure 3 (at the beginning of this proof) shows an example of this correspondence. The top level of Figure 3 shows a string presented to \( N \) with endmarkers. The second level shows the initial and final segments of an accepting computation of \( N \) on the input. The third level shows those arcs in the catenated graph
\[
\begin{align*}
&g(\text{-}) | g(a_1) | g(a_2) | \cdots | g(a_k) | g(\text{-})
\end{align*}
\]
which correspond to steps in the computation. Let us discuss this correspondence.

\( N \)'s first step is right from \( a_1 \) to \( a_2 \). By (1a) in the definition of \( g(a_1) \), arc 1 is present in the catenated graph. Similarly, arc 2 is present by clause (1c), arc 3 by (1d), arc 4 by (2a), and arcs 5, 101, and 102 by (1a). By pursuing this correspondence it can be shown in general that:

**Sublemma 1:** There is a path in \( g(\text{-}) | g(a_1) | g(a_2) | \cdots | g(a_k) | g(\text{-}) \) which correspond to steps in the computation. Let us discuss this correspondence.

\( N \)'s previous move: Moves onto symbol \( g(a_1) \) into state \( q \).

Possible next moves for \( G \): Move one square left, or right, or remain forever on this symbol, according to \( \delta_G(q,g(a_1)) \).

\( D \)'s previous move: Move onto symbol \( a_i \) into state \( q \).

\( D \)'s simulation: imitate the move of \( G \). We set \( \delta_D(q,a_i) = \delta_G(q,g(a_1)) \).

**Proof of (ii), Part 2:**

In Part 1 of the proof we have shown how to encode the computation of any given \( n \)-state non-deterministic machine
\[
N = \langle Q, \Sigma, \rightarrow, I, Q_0, F \rangle
\]
as 2-\( n \)-graphs. Now assume we have a 2-way deterministic finite automaton
\[
G = \langle Q_G, \Gamma, \rightarrow, I, Q_G, F_G \rangle
\]
for the language \( C_G \). \( G \) will be used to construct a deterministic machine \( D \) which is equivalent to \( N \). On input \( x = \text{-}a_1a_2 \cdots a_k \text{-} \), \( D \) will simulate the computation of \( G \) on the input
\[
\text{-}g(\text{-}) g(a_1) g(a_2) \cdots g(a_k) g(\text{-}) \text{-} \).
\]
By the Lemma above, this will cause \( D \) to accept \( x \) iff \( N \) accepts \( x \). We now indicate the correspondence between the computations of \( G \) and \( D \), and define the machine \( D \) by specifying its initial state, final states, and transition function \( \delta_D \). \( D \) will have the same state set as \( G \).

We will analyze the moves of \( G \) in 4 different situations, arranging in each case to have \( D \) simulate the action of \( G \).

(1) Main segment of the computation (cf. Fig. 7).

\[
\begin{align*}
G: & \\
\begin{array}{c|c}
g(a_1) & g(b) \\
\end{array} \\
\downarrow & q \quad \downarrow \quad q' \\
\end{align*}
\]

\[
\begin{align*}
D: & \\
\begin{array}{c}
a_1 \quad \text{b} \\
\end{array} \\
\downarrow & q \quad \downarrow \quad q' \\
\end{align*}
\]

\[
\begin{align*}
\quad & = \text{previous move} \\
\quad & = \text{current move} \\
\end{align*}
\]

Figure 7

G's previous move: Moves onto symbol \( g(a_i) \) into state \( q \).

Possible next moves for \( G \): Move one square left, or right, or remain forever on this symbol, according to \( \delta_G(q,g(a_i)) \).

D's previous move: Move onto symbol \( a_i \) into state \( q \).

D's simulation: imitate the move of \( G \). We set \( \delta_D(q,a_i) = \delta_G(q,g(a_i)) \).

(2) Left endmarker (cf. Fig. 8).

G's previous move: Move left onto the symbol \( g(\text{-}) \) into state \( q \).

Consider \( G \)'s computation on the 2-symbol string
\[
\text{-} g(\text{-}) \text{-}
\]
starting on \( g(\text{-}) \) in state \( q \). Possible outcomes of \( G \)'s subsequent moves:

Case (a): Falls off the right end of the 2-symbol string
\[
\text{-} g(\text{-}) \text{-}
\]
into state \( q' \).

Case (b): Fails to do so, thereby rejecting the input.
D's previous move: Moves left onto the symbol \( \_ \_ \) into state \( q \).

D's simulation:

Case (a): Compress the moves made by \( G \) into a single move. Set \( \delta_D(q, \_ \_ ) = \langle q', R \rangle \).

Case (b): \( G \) has rejected its input. \( D \) should reject also. Set \( \delta_D(q, \_ \_ ) = \phi \).

(3) Right endmarker and final states (cf. Fig. 9).

\[
\begin{array}{c|c|c}
\_ \_ \_ & g(-\_ \_ ) & \_ \\
q & q_1 & q' \\
q_2 & & \\
\end{array}
\]

\[
\begin{array}{c|c|c}
a_k & \_ \_ \_ & \_ \\
q & & q' \\
\end{array}
\]

Make \( q' \) the initial state of \( D \)

Figure 9

G's previous move: Moves right onto \( g(-\_ \_ ) \) in state \( q \).

Outcomes for \( G \): Consider the ensuing computation of \( G \) on the 2-symbol string \( g(-\_ \_ ) \_ \_ \). We distinguish three possible results:

Case (a): \( G \) passes through a final state \( f \), accepting the input.

Case (b): \( G \) does not pass through a final state, and

(i) falls off the left end of \( g(-\_ \_ ) \_ \_ \) into some state \( q' \).

(ii) does not fall off the left end of this string, rejecting the input.

D's previous move: Move right onto \( \_ \_ \) in state \( q \).

D's simulation:

Case (a): Since \( G \) has accepted, \( D \) should accept also. Make \( q \) a final state of \( D \), and set \( \delta_D(q, \_ \_ ) = \phi \).

Case (b):

(i): \( D \), in a single move, simulates \( G \)'s eventual move left into state \( q' \). We set \( \delta_D(q, \_ \_ ) = \langle q', L \rangle \).

(ii): \( G \) has rejected its input. \( D \) should reject also, so we set \( \delta_D(q, \_ \_ ) = \phi \).

(4) Initial states (cf. Fig. 10).

\[
\begin{array}{c|c|c}
\_ \_ \_ & g(-\_ \_ ) & g(a_1) \\
q_1 & q_2 & q' \\
\end{array}
\]

\[
\begin{array}{c|c|c}
a_1 & \_ \_ \_ & \_ \\
q & & q' \\
\end{array}
\]

Make \( q' \) the initial state of \( D \)

Figure 10

G: Begins on \( g(-\_ \_ ) \) in state \( q_G \).

Outcome: Consider the computation of \( G \) on the 2-symbol string \( g(-\_ \_ ) \_ \_ \) starting on \( g(-\_ \_ ) \) in state \( q_G \). If the machine \( G \) is to ever accept anything, it must fall off the right end of this string into a state \( q' \). We want to arrange the initial state of \( D \) to correspond to this situation.

\( D \): Set the initial state of \( D \) to be \( q' \). \( D \) begins on symbol \( a_1 \) in state \( q' \).

This completes our construction of the machine \( D \). In the course of the construction we have considered 4 different kinds of moves made by \( G \), and have arranged in each case to have \( D \) simulate the move. Using this correspondence, it is straightforward to show that \( D \) executes the required simulation of \( G \). This concludes Part 2 of the proof.

\[\text{§2.2 } 1N \rightarrow 2D: \langle B_1, B_2, \ldots \rangle \text{ is COMPLETE}\]

In this section we show that the languages \( B_n \) are complete for \( 1n \rightarrow 2d \) conversion.
Theorem 2.3:
(i): $B_n$ is accepted by an $n$-state 1-way non-deterministic fa.
(ii): Among all languages accepted by $n$-state lnea, $B_n$ requires the largest 2-way deterministic fa.

Proof of (i): Easy. 

Proof of (ii): This proof is similar in structure to the proof of part (ii) of Theorem 2.2. Let $n$-state lnea $N = <Q,G,\delta,Q_0,F>$ be given. Let $G = <Q_G,\Delta,\omega,\epsilon,G,F_G>$ be a 2dfa accepting $B_n$. We will demonstrate a 2dfa $D = <Q,G,\Delta,\omega,\epsilon,\delta,D,Q_0,FD>$ (with the same state set as $G$) such that $D$ accepts $L(N)$.

To each $a \in \Delta$ we associate a graph $g(a) \in G$. In addition, we pick graphs $s,f \in G$ to be associated with the initial and final states of $N$, respectively. We will choose the $g(a), s, f$ to satisfy:

Lemma: $N$ accepts input $a_1a_2 \cdots a_k$ iff the catenated graph $s \mid g(a_1) \mid g(a_2) \mid \cdots \mid g(a_k) \mid f$ is in $B_n$.

Let $q_1,q_2,\ldots,q_n$ enumerate the states of $M$. The Lemma then follows from the Sublemma and (2) and (3) in the definition above.

The proof now proceeds exactly as in Part 2 of the theorem for $C_n$: On input $\omega_a \omega_2 \cdots a_k \omega_f$, 2dfa $D$ is constructed to simulate the computation of $G$ on input $\omega_s \omega_1 \omega_2 \cdots g(a_k) \omega_f$. By the Sublemma, $D$ will be equivalent to $N$. 

§3 AN ANALOGY TO P =? NP

This section presents a convenient notation for comparing the succinctness of description of different models of automata.

Define a language sequence $L$ to be an infinite sequence of languages $<L_1,L_2,...>$. We consider objects of this type because we are interested in the rate of growth of the state complexities of sequences of languages. Given any automata model, say lnea, we wish to classify those language sequences which have succinct representations using this model. Define $IN$ to be the class of language sequences $L = <L_1,L_2,...>$ with the property that some polynomial $p(i)$ bounds the size of the smallest lnea accepting the language $L_i$. Analogously, define the classes ID, 2D, and 2N corresponding to the models 1dfa, 2dfa, and 2nfa. The primary questions we address can now be reformulated as: "Is 2N equal to 2D?" and: "Is $IN$ contained within 2D?"

It is interesting to consider closure properties of these classes. For example, all four classes are closed under union and intersection. By the union of two language sequences $L$ and $L'$ we mean the pairwise union of the component languages $<L_1 \cup L'_1,L_2 \cup L'_2,...>$. Closure under complement is another matter, however. The one way deterministic class, 1D, is clearly closed under complementation, but as we show in the next section, 1N is not closed. We do not know the status of 2D or 2N.

We now consider closure under a certain reducibility.

Definition: For alphabets $\Delta_1$ and $\Delta_2$ and languages $L_1 \subseteq \Delta_1$ and $L_2 \subseteq \Delta_2$, we say that $L_1$ homomorphically reduces to $L_2$ ($L_1 \leq h L_2$), if there is a map $g: \Delta_1 \rightarrow \Delta_2$ and $\ell \in \Delta_2$ such that for any string $s = s_1s_2 \cdots s_k$ ($s \in \Delta_1$), $s \in L_1$ iff $g(s_1)g(s_2) \cdots g(s_k) \ell \in L_2$.

Informally, this says that $L_1$ homomorphically reduces to $L_2$ if by adding end-markers to $L_1$, every string in $L_1$ can be homomorphically mapped to $L_2$. We extend the notion of homomorphic reduction to language sequences. Given two language sequences $L$ and $L'$, say that $L \leq h L'$ if there is a polynomial $p$ such that each $L_i$ is $h$ reducible to $L'_{j}$ for some $j \leq p(i)$.

In this notation, it is the case that for regular languages $L_1$ and $L_2$, if $L_1 \leq h L_2$, then $L_1$ requires a 2dfa at most as large as that required by $L_2$. This can be proved by a generalization of Theorem 2.2 (ii), part 2. From this it follows that 2D is closed under $\leq h$, i.e., for any language sequence $L$ in 2D, if $K \leq h L$ then $K$ is in 2D. Similarly, we can show that 1D, 1N, and 2N are closed under $\leq h$. The language
sequence C has the special property that for every \( L \) in \( 2N \), \( L \leq_h C \). This is a generalization of the lemma to Theorem 2.2 (ii), part 1 stating that if \( L_1 \) is accepted by an \( n \) state 2nfa then \( L_1 \leq_h C_{2n} \). The proof given supports the more general statement.

This discussion indicates a close analogy between the classes \( 2D \) and \( 2N \) and the classes \( P \) and \( NP \). Here, \( \leq_h \) plays the role of polynomial time reducibility and the complete sequence \( C \) corresponds to an \( NP \) complete set such as \( 3SAT \).

Similarly, \( B \) is complete for the class \( IN \) with respect to the \( \leq_h \) reducibility. We now turn our attention to the state complexity of the complete languages \( B_n \) and \( C_n \) under various machine models. We will concentrate on \( B_n \). All of our lower bounds on the size of recognizers for \( B_n \) extend directly to \( C_n \), because a recognizer for \( C_n \) can be converted to one for \( B_n \) by deleting transitions for symbols from \( \Gamma_n \). In some cases, we have tighter lower bounds for \( C_n \).

Let \( s \) be a string over \( \Sigma_n \). Say that \( s \) is live if it is a member of \( B_n \), i.e., if there is a path from any left node to any right node. Otherwise, \( s \) is dead. In addition, node \( m \) in \( s \) is live (dead) means that there is (is not) a path from any left node to the \( m \)th from the top right node in \( s \). Given any ldfa, \( M = \langle Q, \Sigma_n, \delta, q_0, F \rangle \) and arbitrary state \( q \) and input string \( s \), we abbreviate \( \delta(q, s) \) by \( q(s) \) and \( \delta(q_0, s) \) by \( M(s) \).

### §4 LOWER BOUNDS.

We begin by considering deterministic and nondeterministic one way automata. The computations of these machines are relatively easy to analyze and consequently our results are fairly strong.

There is an \( n \) state 1nfa for \( B_n \). Thus by the subset construction there is a \( 2^n \) state 1dfa. The following result shows this to be optimal.

**Theorem 4.1.1**: Any ldfa accepting \( B_n \) has at least \( 2^n \) states.

**Proof**: By counting information. \( \square \)

There is a \( 2^{(n+1)^2} \) state 1dfa for \( B_n \). This is close to optimal.

**Theorem 4.1.2**: Any ldfa accepting \( C_n \) has at least \( 2^{(n-2)^2} \) states.

So we see that nondeterminism is extremely helpful to machines accepting \( B_n \). Curiously, however, it is not of any use at all to machines accepting the complement of \( B_n \), \( \bar{B}_n \).

**Theorem 4.1.3**: Any 1nfa accepting \( \bar{B}_n \) has at least \( 2^n \) states.

**Proof**: Assume to the contrary that \( M \) is a 1nfa accepting \( \bar{B}_n \) with fewer than \( 2^n \) states. For every state \( q \) we wish to let \( r_q \) be the set of nodes which \( M \) thinks are live when it is in state \( q \). Formally, let \( r_q = \{ m \mid \text{for any } y \in \Sigma_n \text{ containing } (m \rightarrow m), q(y) \text{ enters only reject states} \} \). For each \( a \subseteq \{1, n\} \) we let \( x_a \) be \( \{ (1 \rightarrow m) \mid m \in a \} \).

**Fact 1**: For every \( a \subseteq \{1, n\} \) and any \( q \in M(x_a) \), \( a \subseteq r_q \).

Otherwise there would be a live string \( x_ay \) which would be accepted by \( M \), a contradiction.

**Fact 2**: For some \( a \subseteq \{1, n\} \), every \( q \in M(x_a) \) has the property that \( r_q = a \).

Otherwise, each subset \( a \) would have a distinct state \( q \) associated with it, implying that there are at least \( 2^n \) states in \( M \), a contradiction. \( \square \)

Let \( a \) be as in fact 2. Let \( z \in \Sigma_n \) be \( \{(m \rightarrow m) \mid m \in a \} \). The string \( x_aza \) is in \( \bar{B}_n \) yet we claim that \( M \) rejects it. To see this, note that for all \( q \in M(x_a) \), there is an \( m \in r_q - a \) (by facts 1 and 2). For each such \( m \), \( (m \rightarrow m)za \) and thus each \( q \in M(x_a) \) is driven to reject states by \( z \) (by the definition of \( r_q \)). Hence all branches of the computation of \( M \) on \( x_aza \) terminate in reject states. \( \square \)

A corollary to this theorem is that the class \( 1N \) is not closed under complement. This is indicated in the following diagram.

This diagram presents the question of whether \( 1D \) is equal to the intersection of \( 1N \) and \( \text{co-}1N \). The languages \( <(0+1)^{*11(0+1)^{n}} \mid N = 1, 2,... > \) witness a negative answer to this.

We also have enough machinery to easily solve the problem of whether \( 2D \) is contained within \( 1N \). Restrict the graphs of \( \Sigma_n \) to not have any right nodes with more than one arc. The strings over this new alphabet our thus restricted to be forests. Define \( T_n \) to be \( B_n \) restricted to such strings. It is straightforward to accept \( T_n \) with an \( O(n^2) \) state 2dfa, and it is not any harder to accept \( \bar{T}_n \).

However, as a corollary to the above proof, we see that \( \bar{T}_n \) requires \( 2^n \) states on a 1nfa. In fact, by taking the join of \( T_n \) and \( \bar{T}_n \) in some reasonable way, i.e., \( \bar{T}_n = \{ s \mid s \in \bar{T}_n \text{ iff the length of } s \text{ is even} \} \), we obtain languages with succinct 2d acceptors yet neither they nor their complements have succinct 1n descriptions. Thus \( 2D \) is not contained within \( 1N \cup \text{co-}1N \).
§4.2 Restricted two way automata (parallel machines)

Theorems 2.1 and 2.2 reduce the question of whether \( 2^n - 2 \) \((1n - 2d)\) conversion is polynomially bounded to the question of whether for some polynomial \( p(n) \), \( C_n(B_n) \) is accepted by a \( p(n) \) state 1dfa. We conjecture that polynomial conversion is not possible. In order to gain insight into this we consider 2d acceptors whose behavior has been restricted.

Restrictions that are placed on the 2d automata are of two forms: limiting head behavior and limiting communication. We limit the head behavior by permitting the machine to only make a series of one-way passes over the input. The communication is limited by restricting the exchange of information between sweeps. One version of this is the parallel union finite automaton.

Definition: A **parallel union finite automaton (pufa)** \( P \) is a set \( \{M_1, \ldots, M_k\} \) of 1dfa. The language accepted by \( P \) is the union of the languages accepted by its component machines.

For some languages, pufa can be exponentially more succinct than 1dfa. For example \( \{x \neq y \mid x, y \in \{0, 1\}^n \} \) and \( x \neq y \) is the union of \( n \) \( O(n) \) state 1dfa, yet requires a \( 2^n \) state 1dfa. However, we can show that pufa are not more succinct than 1dfa for \( B_n \).

**Theorem 4.2.1:** In any parallel union finite automaton accepting \( B_n \), one of the component machines must have at least \( 2^n \) states.

**Proof sketch:** Assume to the contrary that there is a pufa accepting \( B_n \), and containing component machines all with fewer than \( 2^n \) states. Choose any component machine. Having too few states to distinguish all possible subsets of the \( n \) nodes, it occasionally gets "confused" as to whether some node is actually live. The key point is that whenever the machine is uncertain as to the status of a particular node, it must assume that it dead. Otherwise, if it were to wrongly assume that the node was live and accepted the input based upon that assumption, then the entire pufa would wrongly accept. Knowing this, our procedure is to construct a string in which some node is live yet which fools this component machine into assuming that it is dead. We then extend the string, continuing the path from that node in such a way as to fool a second component machine. Ultimately we get a live string which fools all of the component machines into believing it is dead. This string causes the pufa to err.

A more formal proof follows.

**Proof:** We perform an induction on \( k \), the number of machines in \( P \).

**Basis,** \( k = 1 \):

This follows from theorem 4.1.1, since a pufa with only one component machine is in fact a 1dfa.

**Induction,** proving case \( k \) from case \( k - 1 \):

Our induction hypothesis is that the theorem holds for pufa having fewer than \( k \) components. Suppose the theorem fails for pufa \( P = \{M_1, \ldots, M_k\} \) all of whose components have fewer than \( 2^n \) states. In particular, component machine \( M_1 \) has fewer than \( 2^n \) states. For every state \( q \) of \( M_1 \), we say that \( q \) is **dead** if there is no string \( s \) which drives \( q \) to an accept state, otherwise we say that \( q \) is **live**. Any string which drives \( M_1 \) to a dead state is said to **kill** \( M_1 \).

**Fact 1:** Any dead string \( s \) kills \( M_1 \).

Otherwise \( s \) drives \( M_1 \) to a live state \( q \) which then can be driven via some string \( t \) to an accept state. The string \( st \) is dead yet \( M_1 \) and therefore \( P \) accepts it, a contradiction. \( \diamond \)

**Fact 2:** There is a live string \( v \) which kills \( M_1 \).

Otherwise, all live strings drive \( M_1 \) to live states. On the other hand, fact 1 states that all dead strings drive \( M_1 \) to dead states. By then designating all the live states of \( M_1 \) to be accept states and all the dead states to be reject states, we obtain a 1dfa accepting exactly the live strings. This machine recognizes \( B_n \) with fewer than \( 2^n \) states, contradicting theorem 4.1.1. \( \diamond \)

Let \( r \) (reset) be the complete bipartite graph on \( 2^n \) nodes, i.e., \( r = \{(i-j) \mid i, j \in [1,n]\} \) and let \( u = vr \).

**Example:** \( r \) with \( n = 4 \)

For each component machine \( M_i \), define a new machine \( M'_i \) obtained by changing the start state of \( M_i \) to be \( M_i(u) \). Let \( R_i \) and \( R'_i \) be the languages accepted respectively by the machines \( M_i \) and \( M'_i \). By assumption \( P \) accepts \( B_n \); in other words \( B_n = R_1 \cup \cdots \cup R'_k \). Since \( \{t \mid t \in B_n\} = B_n \), \( B_n = R'_1 \cup \cdots \cup R'_k \). However, \( u \) kills \( M_1 \) and thus \( R'_1 = \emptyset \). This allows us to conclude that \( B_n = R'_2 \cup \cdots \cup R'_k \).

Now we construct a pufa \( P' = \{M'_2, \ldots, M'_k\} \) which accepts \( B_n \) and yet has only \( k-1 \) machines, all with fewer than \( 2^n \) states. This contradicts the induction hypothesis. \( \square \)

**Remarks:** The existing techniques for proving lower bounds by counting information vs. crossing sequences are insufficient to give this result, as it is possible for a large number of small machines to have enough states among them to carry the necessary information across. This result shows that their inability to communicate prevents them from successfully doing so. Also, this proof provides a bound of **exactly** \( 2^n \) states. Thus we know that the best pufa is actually the obvious single \( 2^n \) state 1dfa.

A similar proof shows:

**Theorem 4.2.2:** In any pufa accepting \( C_n \), one of the component machines must have at least \( 2^{n-2} \) states.

Definition: A **parallel intersection finite automaton (pi/fa)** \( P \) is a set \( \{M_1, \ldots, M_k\} \) of 1dfa. The language accepted by \( P \) is the intersection of the languages accepted by its component machines \( M_i \).
Theorem 4.2.3: If \( P = \{ M_1, \ldots, M_k \} \) is a pifa accepting \( B_n \), then one of its component machines has at least \( 2^n \) states.

Proof: Assume that a pifa \( P \) exists which contradicts the theorem. We further assume wlog that each component machine has at least one reject state. Our goal is to construct a string \( sB_n \) which deceives \( P \) into accepting it. The construction is in stages, the \( i^{th} \) stage deceiving \( M_i \). This is done by convincing \( M_i \) that some node is live when in fact it isn't. That node is then connected to the end of the string, causing \( M_i \) to wrongly accept.

Three variables are used: \( l \) denotes the current live node, \( d \) denotes the node which the current machine believes is live but which actually is dead, and \( s \) denotes the current partial string.

Stage 0: Let \( s_0 = ((1 \ldots 1)) \), \( l_0 = 1 \), and \( d_0 = 2 \).

Stage \( i \): We are assuming from the previous stage that in string \( s_{i-1} \), node \( l_{i-1} \) is live and node \( d_{i-1} \) is dead. Our goal for this stage is to preserve these properties for \( s_i \), \( l_i \), and \( d_i \). In addition, we must connect node \( d_i \) to node \( d_{i-1} \) of the previous stage.

Consider \( M_i \). For each state \( q \) of \( M_i \), we let \( r_q \) be the set of nodes which \( M_i \) believes are live when it is in state \( q \). Formally, \( r_q = \{ m \mid \text{all strings which contain a path from left node } m \text{ to any right node, drive } q \text{ to an accept state} \} \). For every \( a \subseteq [1,n] \) let \( x_{a}eS_n \) be the graph \( \{(d_{i-1}m) \mid me[a,1,n]\} \cup \{(l_{i-1}m) \mid mea\} \)

Example: \( x_{a} \) with \( n = 4 \), \( d_{i-1} = 2 \), \( l_{i-1} = 4 \), and \( a = \{3,4\} \)

Fact 1: For every \( a \subseteq [1,n] \), the set of live nodes of \( s_{i-1}x_{a} \) is \( a \).

This follows from the stated properties of \( s_{i-1}, l_{i-1}, \) and \( d_{i-1} \).

Fact 2: For every \( a \subseteq [1,n] \), if \( M_i(s_{i-1}x_{a}) = q \) then \( a \subseteq r_q \).

Otherwise, suppose there is some \( m \in a - r_q \). By fact 1 and the definition of \( r_q \), there is some \( t \) such that \( M_i(s_{i-1}x_{a}t) \) is rejecting yet \( s_{i-1}x_{a}t \) is in \( B_n \), a contradiction.

Fact 3: There are 3 distinct, nonempty subsets \( a, b \subseteq [1,n] \) and a state \( q \) such that \( M_i(s_{i-1}x_a) = M_i(s_{i-1}x_b) = q \).

This is true because, for all nonempty subsets \( a, s\ldots x_a \) is live (by fact 1) and therefore \( M_i(s_{i-1}x_a) \) is accepting. There are \( 2^n-1 \) non empty subsets and at most \( 2^n-2 \) accepting states, thus by counting, the desired \( a \) and \( b \) exist.

Let \( a, b, \) and \( q \) be as in fact 3. Since \( a \) and \( b \) are distinct, we assume wlog that there is some \( m \in b - a \). By fact 2 we know that \( a, b \subseteq r_q \) and thus \( m \in r_q \). Let \( s_i = s_{i-1}x_a, d_i = m, \) and \( l_i = \) any node in \( a \).

Verifying the desired relationships between \( s_i, d_i, \) and \( l_i \), we see by fact 1 that node \( d_i \) is dead because \( d_i \in a \) and \( l_i \) is live because \( l_i \in a \). Furthermore, \( d_i \) is connected to \( d_{i-1} \) of the previous stage. Subsequent stages will ensure that \( d_i \) is connected to a right node of \( s \) and therefore, since \( d_i \in r_q \), \( M_i \) will accept \( s \).

End of stage \( i \).

Finally we let \( s = s_iy \) where \( y = \{(d_i-1)\} \). Straightforward inductions show that for each stage the node at \( l_i \) connects to the start but not the end of \( s \), and the node at \( d_i \) connects to the end but not the start. From this we conclude that \( sB_n \) and that each \( M_i \) accepts \( s \), a contradiction.

A similar proof shows that:

Theorem 4.2.4: In any pifa accepting \( C_n \), one of the component machines must have at least \( 2^{(n-3)^2} \) states.

§4.3 Restricted two way automata (series machines)

The automata considered in the previous section are very limited in the sense that no communication is permitted between the component machines. In the next model under consideration, the \( i^{th} \) component machine \( M_i \) may pass information to the \( i+1^{st} \) machine. This done by using the result of the computation of \( M_i \) to determine the starting state of \( M_{i+1} \).

Definition: A series finite automaton (sfa) \( S \) is an ordered collection of ldfs, \( (M_1, \ldots, M_k) \) together with functions \( \{ f_1, \ldots, f_{k-1} \} \) where each \( f_i \) is a mapping of the states of \( M_i \) into the states of \( M_{i+1} \). On input \( x \), \( S \) runs these machines one at a time, in order, over the input tape. The ending state of \( M_i \) on \( x \) determines, via \( f_i \), the starting state of \( M_{i+1} \). The final state of \( M_k \) determines the acceptance of \( x \).

These machines are more complex than parallel machines and correspondingly, our results are weaker. Even the case where the sfa has only two components appears difficult to analyze. We conjecture that in this case, one of the component machines must have at least \( 2^n \) states in order that the series fa accept \( B_n \). The main result of this section is in support of this.

Theorem: Given \( S \), a series automaton accepting \( B_n \) with only two component machines \( M_1 \) and \( M_2 \). If either component has fewer than \( \sqrt{2^{n-1}} \) states, then the other one has at least \( 2^n \) states. The square root function in our discussions always rounds up to an integer.

The proof of this depends upon an analysis of the degree to which a ldfa can be "confused", as a function of the number of states it contains. We formalize this notion of confusion as follows.

Given any string \( s \) over \( \Sigma_n \), we define the accessibility set of \( s \) to be the set of live nodes of \( s \). Call the two strings equivalent if their accessibility sets are equal. Let \( q \) be any state in a ldfa \( M \) which operates over \( \Sigma_n \). We say that accessibility set \( a \) is associated with state \( q \) if there is some string
with accessibility set $a$ which drives $M$ to $q$. The state $q$ is
called $k$ confused if there are at least $k$ different accessibility
sets associated with it. Machine $M$ is $k$ confused if it has a $k$
confused state. For example, the $2^n$ state acceptor for $B_n$
is only 1 confused, and it follows from lemma 1, that any
machine with fewer than $2^n$ states is at least 2 confused.
Trivially, the 1 state ldfa is 2 confused. It seems reasonable
to expect that the smaller a machine is, the more confused
it must get. The following table summarizes or knowledge of this phenomenon.

<table>
<thead>
<tr>
<th># states</th>
<th>degree of confusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt; n$</td>
<td>total ($2^n$) confusion</td>
</tr>
<tr>
<td>$\geq n$</td>
<td>$&lt; 2^n$ confusion possible</td>
</tr>
<tr>
<td>$\leq 2^n - 2$</td>
<td>$\geq \sqrt[2^n]{2^{n-1}}$ confusion</td>
</tr>
<tr>
<td>$\leq 2^n - 1$</td>
<td>2 confusion</td>
</tr>
<tr>
<td>$\geq 2^n$</td>
<td>1 confusion possible</td>
</tr>
</tbody>
</table>

Of the following four theorems, 4.3.1 through 4.3.4, only the
last is requisite to the main result of this section. The others
present related aspects of confusion.

**Theorem 4.3.1:** There is a $2^n - 1$ state ldfa which is not more
than 2 confusable.

**Proof:** We construct $M$ as follows. Each state of $M$
is assigned one of the $2^n - 1$ nonempty accessibility sets. As long
as the input string is live, $M$ has no trouble staying in the
assigned state. However, there are no states left to assign to the
empty accessibility set and therefore the dead strings must drive
$M$ into the same states as do the live ones. It follows that there are two accessibility sets associated with each state,
namely one nonempty set and the empty set. Hence each
state is 2 confused and $M$ is only 2 confused. □

For the next theorem, it is useful to note that there is a
two element subset of $\Sigma_n$ which preserves all of the descriptive
power. Let $\Sigma_n' = \{ p_1, p_2, x_n, x_d \}$ where

- $p_1 = [(1-2), (2-1)] \cup [(i-1) | i \epsilon [3, n)]$
- $p_2 = [(i-j) | j = i+1 \ mod n]$
- $x_n = [(i-1) | i \epsilon [1, n]) \cup [(1-2)]$
- $x_d = [(i-i) | i \epsilon [2, n)]$

The languages obtained by restricting $B_n$ to strings over $\Sigma_n'$
still possess the completeness property.

**Theorem 4.3.2:** There is an $n$ state ldfa which is not $2^n$
confusable.

**Proof:** Let $q_1, \ldots, q_n$ be the states of ldfa $M$. $M$ will be
constructed so that every accessibility set is associated with $q_i$
except for the singleton $\{ i \}$. We only need to define $M$ over the restricted alphabet $\Sigma_n' = \{ p_1, p_2, x_n, x_d \}$ discussed above.

 Arbitrarily, we let $M$ start in $q_1$. If $M$ is in $q_i$ and reads
input symbol $p_1$ or $p_2$ then $M$ enters $q_j$, where $j$ is the node
that $i$ is carried to under the permutation. If $M$ is in $q_i$ and
reads either $x_n$ or $x_d$ then $M$ enters $q_1$. □

**Theorem 4.3.3:** Any ldfa with fewer than $n$ states is $2^n$
confusable.

**Proof:** The intuition behind this proof is that partial confusion
is used to induce greater confusion. We are given a ldfa $M$
with $k$ states, where $k < n$. We will focus our attention
upon certain accessibility sets, namely the singletons: $\{ 1 \}, \{ 2 \}, \ldots, \{ n \}$. First, we show that $M$ get confused on
the singletons alone.

**Fact:** There are $k$ singletons which are all associated
with the same state.

We prove this fact inductively by showing that for every
$i \leq k$, there is some set, $A_i$, of $i$ singletons and a state $q_i$
with which all are associated.

**Basis,** $i = 1$:

Trivially true since the singleton $\{ 1 \}$ must be associated with
some state $q_1$.

**Induction,** proving case $i+1$ from case $i$.

The induction hypothesis gives us a set $A_i$, containing $i$
singletons and a state $q_i$ such that every singleton in $A_i$
is associated with $q_i$. We first show that every set of $i$
singletons can be assigned a state $q'$ such that each member of the set is associated $q'$. Choose any set $A'$ containing $i$ singletons. Say $A'_i = \{ m_1, \ldots, \{ m_k \} \}$ and $A' = \{ \{ m_1 \}, \ldots, \{ m_k \} \}$. Let $x \in \Sigma_n$
be the graph $\{ (m_j \rightarrow m_j') \ | \ j \epsilon [1, n] \}$. It is not hard to see that
$q_i(x)$ is the state $q'$ that we wish to assign to $A'$. Thus every
set $A'$ can be assigned a state $q'$.

Now we show that some state $q$ is assigned to two distinct sets, $A'$ and $A''$. This is because there are more sets
than states, i.e., there are $\binom{n}{i}$ different sets of $i$ singletons and
since $1 \leq i \leq n$, we know that $\binom{n}{i} > n - k$ = the number
of states. Hence $q$ exists.

Finally we construct $A_{i+1}$ and $q_{i+1}$. The set $A'$ and $A''$
each contain $i$ singletons thus between them there must be at least
$i+1$ singletons. Since both $A'$ and $A''$ were assigned to $q_i$
each of these $i+1$ singletons is associated with $q_i$. Consequently,
we let these singletons constitute $A_{i+1}$ and we let $q$ be
$q_{i+1}$.

**Fact proved.**

Let $m_1, \ldots, m_k$ and $q_{k}$ be the singletons and state
shown to exist by the above fact. If $M$ is not $2^n$ confusable
then for each state $q_i$ there is some accessibility set, $b_j$,
which is not associated with $q_i$. Let $x \in \Sigma_n$
be $\{ (m_j \rightarrow m) \ | \ m \epsilon [1, k] \}$. Let $q = q_k(x)$. Since each $\{ m_j \}$
is associated with $q_k$, each $b_j$ is associated with $q_i$. However,$b_j$ is not associated with $q_i$ for every $j$, and so $q$ is not equal
to any $q_j$, a contradiction. □

The following result is used in theorem 4.3.5.

**Theorem 4.3.4:** Any ldfa with fewer than $2^n - 1$ states is at
least $\sqrt{2^{n-1}}$ confusable.

**Proof:** Given $M$ with fewer than $2^n - 1$ states. As a first step
we demonstrate that there are two accessibility sets $a$ and $b$
associated with the same state, where $a \notin b$ and $b \notin a$. Since $M$
has fewer than $2^n-1$ states and there are $2^n-1$ nonempty accessibility sets, there must be two distinct sets, $a'$ and $b'$, which are associated with the same state $q$. If $a'$ and $b'$ meet the desired conditions for $a$ and $b$ we are done, otherwise assume wlog that $a \subseteq b$. Let $i$ be a node in $a$ and $j$ a node in $b-a$. For each pair of accessibility sets $c$ and $d$ where $c \subseteq d$, we assign that pair to some state $r$ with which they are both associated, as follows. Let $x \Sigma_\alpha$ be
\[(i-m) | mec \} \cup (j-m) | med \}.
\]

It is not hard to see that $q'(x)$ is the desired state $r$.

There are $3^n-2^n$ such pairs and fewer than $2^n$ states. Thus there must be a state to which at least $(3^n-2^n)/2^n = (3/2)^n-1$ pairs are assigned. From this we can conclude that there must be at least $\sqrt{(3/2)^n-1}$ accessibility sets associated with that state. For sufficiently large $n$ (see note following proof) $\sqrt{(3/2)^n-1} > n+1$ and thus we have at least $n+2$ distinct sets associated with that state. However there cannot exist a chain $a_1 \subseteq a_2 \subseteq \ldots \subseteq a_{n+2}$ of length $n+2$ if each of the $a_i$ is a subset of $[1,n]$. Hence, there must be two accessibility sets, $a$ and $b$, associated with the same state, such that $a \not\subseteq b$ and $b \not\subseteq a$. This completes the first step of the proof.

Now we essentially repeat this idea using the sets $a$ and $b$ rather than $a'$ and $b'$. Choose $i \in a-b$ and $j \in b-a$ and define $x$ as before, except we now permute arbitrary pairs of sets $c$ and $d$. Again, each pair is assigned a state with which each member of the pair is associated. This time there are $2^n(2^n-1)/2$ pairs. Since there are fewer than $2^n$ states, at least $2^n-1$ pairs are assigned to the same state, implying that there must be at least $\sqrt{2^{n-1}}$ accessibility sets associated with that state.

\[\square\]

\textbf{Note:} We have assumed in the first step of this proof that we are dealing with sufficiently large $n$. Nevertheless, the theorem holds for all $n$ by examining the small $n$ individually and applying more careful counting arguments. The details are omitted.

\begin{theorem}
Given $S$, a series automaton accepting $B_n$ with only 2 component machines $M_1$ and $M_2$. If either component has fewer than $\sqrt{2^{n-1}}$ states, then the other one has at least $2^n$ states.
\end{theorem}

\begin{proof}
We will actually prove that the other component has at least $2^n-1$ states, a slightly weaker result. The stronger result holds by an additional argument, which we omit.

\textbf{Proof:} Suppose $s \in S$ violates these conditions. Our plan is to construct a live string $t_L$ and a dead string $t_D$, which are jointly accepted or rejected by $S$. These strings will be constructed in three parts; a common live beginning string $s_L$, a common live ending string $s_D$, and middle strings $s_L$ and $s_D$. That is, $t_L$ will be $s_L s_D s_E$ and $t_D$ will be $s_D s_L s_E$. The actual deception of $P$ occurs during the middle strings. The beginning and ending strings are merely to enable us to predict the starting state of $M_2$.

First we analyze the structure of $M_1$. Call a string $q$ \textit{internal} if for every live $s$ there is a string $t$ such that $st$ is live and $q(st) = q$. Let $x \Sigma_\alpha$ be the complete bipartite graph \[(i-j) \mid i,j \in [1,n] \}.
\]

\textbf{Fact 1:} There is some live string $s$ which drives $M_1$ to an internal state.

If the starting state $q_0$ of $M_1$ is internal then we are done, otherwise there is some live string $s_1$ such that any live extension of $s_1$ will not carry $M_1$ back to $q_0$. Let $q_1 = q_0(s_1 x)$. This means that $q_0$ is unreachable from $s_1$ via live strings. If $q_1$ is not internal we can repeat the above to obtain a new state $q_2$ from which it is impossible to reach either $q_0$ or $q_1$ via live strings. Since $M_1$ is finite state this process cannot go on forever and so eventually we will obtain an internal state $q_L$. The string $s$ which drives $M$ to $q_L$ is $s_1 s_2 x \ldots s_n x$.

We let $s_E$ be the string $s$ constructed in fact 1.

Our next step is to construct the strings $s_L$ and $s_D$. For this step we need to know the starting state of $M_2$ which depends upon the ending state of $M_1$ on inputs $t_D$ and $t_L$ which in turn depend upon the strings we are currently constructing. To get around this circularity we assume for this step that $M_2(t_L) = M_2(t_D) = M_2(s_E)$, the internal state of fact 1. This determines $r_0$, the starting state of $M_2$. The construction of $s_E$ in the final step of our proof ensures that this assumption is valid. We let $q_1$ and $r_1$ be $M_1(s_D)$ and $r_0(s_D)$ respectively. We assume wlog that $M_2$ has fewer than $\sqrt{2^{n-1}}$ states, and thus $M_2$ has fewer than $2^n-1$ states.

Consider the automaton $M_2$ obtained by changing the start state of $M_2$ to $q_1$. Since $M_2$ has fewer than $2^n-1$ states, $M_2$ is $\sqrt{2^{n-1}}$ confusable (by theorem 4.3.4). In other words, there are $\sqrt{2^{n-1}}$ pairwise inequivalent strings, $s_1, \ldots, s_k$, where $k = \sqrt{2^{n-1}}$, which all drive $M_2$ from state $q_1$ to the same state $q$. Now $M_2$ has fewer than $\sqrt{2^{n-1}}$ states, thus two of these strings, $s_j$ and $s_k$, must drive $M_2$ from $r_1$ to the same state $r$. Since $s_j$ and $s_k$ are inequivalent, there is some node $m$ which is live in, say $s_j$, and dead in $s_k$. We adjoin the symbol $x = (m(n-1))$ to both strings obtaining a live string $s_j$ and a dead string $s_k$. Let state $q_2$ and $r_2$ be states $q(x)$ and $r(x)$. Note that both $s_L$ and $s_D$ drive $M_1$ from $q_1$ to $q_2$ and $M_2$ from $r_1$ to $r_2$.

At this point, all that remains is to construct $s_E$, a live string which drives $M_1$ from $q_2$ back to $q_1$. This is easy since $q_1$ is an internal state and thus the live string $s_j$ can be extended to a live string $s_L s_E$ which drives $q_1$ to $q_1$.

\[\text{Example: } x \text{ with } n = 5, i = 2, j = 4, c = \{2, 3\}, d = \{1, 2, 3, 5\}\]
So, no matter whether $S$ receives $t_L$ or $t_D$ as input, $M_2$ will be driven into the same state, a contradiction.

§5 RELATED WORK

Meyer and Fischer [8] first considered the relative succinctness of various kinds of descriptions of regular sets.

Joel Seiferas [9] has investigated $1n\rightarrow 2d$ conversion. He demonstrates lower bounds for a restricted 2dfa model and considers several interesting regular languages. For example, let alphabet $\Delta_n$ be the power set of $\{1, \ldots, n\}$ and let $L_n$ be:

$$\bigcup_{x \in \Delta_n} a \ (\Delta_n)^*$$

We can show $L_n$ to be complete for $1n\rightarrow 2d$ conversion by demonstrating that for any $n$ state 1n language $K$, $K \le_1 L_n$.

Acknowledgements

This research began with a study of the language $B_n$. We wish to thank Dana Angluin for posing the questions which initiated this study.

We are deeply indebted to Manuel Blum for many valuable suggestions.

REFERENCES


