Theoretical Computer Science 13 (1981) 323–330
North-Holland Publishing Company

NOTE

SUCCINCT REPRESENTATION OF REGULAR LANGUAGES BY BOOLEAN AUTOMATA*

Ernst LEISS**
Department of Computer Science, University of Kentucky, Lexington, KY 40506, U.S.A.

Communicated by M. Nivat
Received August 1979
Revised May 1980

Abstract. Boolean automata are a generalization of finite automata in the sense that the 'next state', i.e. the result of the transition function given a state and a letter, is not just a single state (deterministic automata) or a union of states (nondeterministic automata) but a boolean function of states. Boolean automata accept precisely regular languages; furthermore they correspond in a natural way to certain language equations as well as to sequential networks. We investigate the succinctness of representing regular languages by boolean automata. In particular, we show that for every deterministic automaton $\boldsymbol{A}$ with m states there exists a boolean automaton with $\lceil \log_2 m \rceil$ states which accepts the reverse of the language accepted by $\boldsymbol{A}$ ($m \geq 1$). We also show that for every $n \geq 1$ there exists a boolean automaton with n states such that the smallest deterministic automaton accepting the same language has $2^{(2^n)}$ states; moreover this holds for an alphabet with only two letters.

1. Notation

We will review some concepts central to this paper; the undefined notions can be found in any standard text book covering finite automata and regular languages.

A boolean automaton ([2]; for more details and proofs see there) is a quintuple $\boldsymbol{B} = (A, Q, \tau, f^0, F)$ where A is the input alphabet, Q is the finite nonempty set of states, $\tau : Q \times A \to B_Q$ is the transition function, B_Q denoting the free boolean algebra generated by Q, $f^0 \in B_Q$ is the initial function, and $F \subseteq Q$ is the set of final states. The operations of B_Q are $\cup$ (union), $\cap$ (intersection), and $^-$ (complement); note that Q is the set of states of $\boldsymbol{B}$ and at the same time the set of generators of B_Q. The transition function τ is extended to $B_Q \times A^*$ as follows. Let $Q = \{q_1, \ldots, q_n\}$. For all $w \in A^*$, $a_j \in A$, $q_i \in Q$, $f \in B_Q$ we define $\tau(q_i, \lambda) = q_i$,[1] $\tau(q_i, a_j w) = f_{ij}(\tau(q_1, w), \ldots, \tau(q_n, w))$, where $f_{ij}(q_1, \ldots, q_n) = \tau(q_i, a_j) \in B_Q$, and $\tau(f, w) = f(\tau(q_1, w), \ldots, \tau(q_n, w))$. We define a relation $=_F$, called evaluation under F, as

* Part of this research was done while the author was visiting the Departmento de Ciencias de la Computación, Universidad de Chile, Santiago, Chile, supported by a grant from the IBM World Trade Americas/Far East Corporation.

** Present address: Department of Computer Science, University of Houston, Houston, TX 77004, U.S.A.

[1] λ denotes the empty word.

follows: Let $\delta_i = 1$ if $q_i \in F$, otherwise $\delta_i = 0$. For any $f \in B_Q$, $f =_F \alpha$ iff $f(\delta_1, \ldots, \delta_n) = \alpha$, $\alpha \in \{0, 1\}$. A word w is accepted by $\boldsymbol{B}$ iff $\tau(f^0, w) =_F 1$. The set of all such words is $L(\boldsymbol{B})$. – Recently we learned that boolean automata have been introduced by Kozen [3] under a different name and in a different context.

Intuitively a boolean automaton can be viewed as a parallel finite automaton (its name in [3]) as a letter of the alphabet is applied (in parallel) to each state and then the results are combined in a boolean function.

If $f^0 \in Q$, $\tau(q_i, a_j) \in Q$ for all i, j, then $\boldsymbol{B}$ is a deterministic (finite) automaton; if $f^0 \subseteq Q$, $\tau(q_i, a_j) \subseteq Q$ for all i, j, then $\boldsymbol{B}$ is a nondeterministic automaton. It is easily verified that these are precisely the usual definitions. We will always assume that our boolean automata are connected i.e. that for no $P \subsetneq Q$, $\{\tau(f^0, w) \mid w \in A^*\}$ is a subset of $B_P \subsetneq B_Q$. This clearly extends the definition of connected finite automata.

It is known ([2, 3]) that $L(\boldsymbol{B})$ is a regular language for any boolean automaton $\boldsymbol{B}$. Furthermore the derived deterministic automaton $\boldsymbol{A}_{\boldsymbol{B}}$ accepts exactly $L(\boldsymbol{B})$; $\boldsymbol{A}_{\boldsymbol{B}}$ is defined as follows: $\boldsymbol{A}_{\boldsymbol{B}} = (A, P, \mu, f^0, G)$, where $P = \{\tau(f^0, w) \mid w \in A^*\} \subseteq B_Q$, $G = \{f \in P \mid f =_F 1\}$, and $\mu(\tau(f^0, w), a) = \tau(f^0, wa)$ for all $w \in A^*$, $a \in A$. Clearly, if $\boldsymbol{B}$ has n states, $\boldsymbol{A}_{\boldsymbol{B}}$ can have no more than $2^{(2^n)}$ states.

The reverse $\boldsymbol{A}^\rho$ of a (connected) deterministic finite automaton $\boldsymbol{A} = (A, Q, \tau, q_0, F)$ is defined as follows. For any $w \in A^*$ let $Q_w = \{q \in Q \mid \tau(q, w) \in F\}$. Then $\boldsymbol{A}^\rho = (A, P, \mu, p_0, G)$, where $P = \{p \mid p = Q_w \text{ for some } w \in A^*\}$, $p_0 = F$, $G = \{p \in P \mid q_0 \in p\}$ and $\mu(p, a) = \{q \in Q \mid \tau(q, a) \in p\}$ for $p \in P$, $a \in A$. $\boldsymbol{A}^\rho$ is always reduced, has at most 2^n states if $\boldsymbol{A}$ has n states, and the language accepted by $\boldsymbol{A}^\rho$ is precisely the reverse of the language accepted by $\boldsymbol{A}$, $(L(\boldsymbol{A}))^\rho = L(\boldsymbol{A}^\rho)$ (see [1]).

2. The results

In [2] systems of left-language equations were studied; these are equations of the form

$$X_i = \bigcup_{a \in A} a \cdot F_{i,a}(X_1, \ldots, X_n) \cup \delta_i, \quad i = 1, \ldots, n,$$

where $F_{i,a}$ is a boolean function in the variables $X_1, \ldots, X_n$, $\delta_i \in \{\{\lambda\}, \emptyset\}$. These equations give rise to a boolean automaton in an obvious way ($\tau(q_i, a)$ corresponds to $F_{i,a}$). In the present note we study the conciseness of this representation of regular languages.

Given a regular language R, we define the (deterministic) complexity of R to be the 'size', i.e. the number of states, of the (uniquely determined) reduced automaton accepting R. It is known that for any $n \geq 1$ there exists a nondeterministic automaton $\boldsymbol{N}$ with n states such that the complexity of $L(\boldsymbol{N})$ is 2^n, i.e. the reduced automaton for the language accepted by $\boldsymbol{N}$ has 2^n states. Moreover this holds for alphabets with a number of letters independent of n (see also Corollary 4). On the other hand, it is also known that for every $n \geq 1$ there exists a language of complexity n such that the smallest nondeterministic automaton has as many states as the reduced deterministic automaton, namely n. Such a class is for instance given by $0^{n-1}0^*$, $n \geq 1$, over the

one-letter alphabet {0}. In this paper we study similar questions in the case of boolean automata.

We first show that the reverse of a regular language of complexity m can be succinctly represented by a boolean automaton.

Theorem 1. *Let $\boldsymbol{A}$ be a deterministic finite automaton with m states. There exists a boolean automaton $\boldsymbol{B}$ with $\lceil \log_2 m \rceil$ states which accepts the reverse language, $L(\boldsymbol{B}) = (L(\boldsymbol{A}))^o$.*

Proof. Let $\boldsymbol{A} = (A, \{0, \ldots, m-1\}, \tau, q_0, F)$, and let $\hat{\imath}$ be the binary representation of the number i with $k = \lceil \log_2 m \rceil$ digits. For example, if $m = 9$, $i = 3$, then $\hat{\imath} = 0011$. Now we can write the transition table of $\boldsymbol{A}$ induced by τ in this notation, i.e. $\hat{\imath}$ under a is $\widehat{\tau(i, a)}$. This can be conceived as the transition table of a sequential network $\boldsymbol{N}$ with decoded inputs where the jth digit of the binary representation corresponds to the jth variable of $\boldsymbol{N}$; clearly, $\boldsymbol{N}$ has k variables.

It is known that the language $L(\boldsymbol{N})$ defined by the network $\boldsymbol{N}$ is exactly $L(\boldsymbol{A})$ (see [2], and the following example). We associate variable X_j with the jth variable of $\boldsymbol{N}$, $j = 1, \ldots, k$, and derive the next-state equations, the output equation, and the starting state of $\boldsymbol{N}$; this is a complete description of our sequential network with decoded inputs. From it we obtain the system of right-language equations whose solution is precisely $L(\boldsymbol{A}) = L(\boldsymbol{N})$ (again see [2]). We get a system of left-language equations whose solution is the reverse of $L(\boldsymbol{A})$ by writing the letters in the right-language equations on the left side [2]. This system of left-language equations can be rewritten as a boolean automaton $\boldsymbol{B}$. It follows that $\boldsymbol{B}$ has $\lceil \log_2 m \rceil$ states, and $L(\boldsymbol{B}) = (L(\boldsymbol{A}))^o$; hence $\boldsymbol{B}$ is the desired boolean automaton.

We give a detailed example. Let $\boldsymbol{A} = (\{a, b, c\}, \{0, 1, 2, 3\}, \tau, 0, \{0\})$ be the given deterministic automaton, τ being defined by

	a	b	c
0	1	1	3
1	2	0	1
2	3	2	2
3	0	3	3

We rewrite this in binary notation and obtain the transition table of a sequential network with decoded inputs:

		a		b		c		
		Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	z
y_1	y_2							
→0	0	0	1	0	1	1	1	1
0	1	1	0	0	0	0	1	0
1	0	1	1	1	0	1	0	0
1	1	0	0	1	1	1	1	0

where the arrow ($\rightarrow$) indicates the starting state (00) and the last column (z) defines the output equation ($z = \bar{y}_1 \cap \bar{y}_2$). From the transition table we obtain the next-state equations, the output equation, and the starting state:

$$Y_1 = a \cap (y_1 \cap \bar{y}_2 \cup \bar{y}_1 \cap y_2) \cup b \cap (y_1) \cup c \cap (y_1 \cup \bar{y}_2),$$

$$Y_2 = a \cap (\bar{y}_2) \cup b \cap (y_1 \cap y_2 \cup \bar{y}_1 \cap \bar{y}_2) \cup c \cap (\bar{y}_1 \cup y_2),$$

$z = \bar{y}_1 \cap \bar{y}_2$ output function,

(0, 0) starting state.

This network describes exactly $L(\boldsymbol{A})$. Now we get the right-language equations which also describe $L(\boldsymbol{A})$:

$$x_1 = (x_1 \cap \bar{x}_2 \cup \bar{x}_1 \cap x_2) \cdot a \cup (x_1) \cdot b \cup (x_1 \cup \bar{x}_2) \cdot c,$$

$$x_2 = (\bar{x}_2) \cdot a \cup (x_1 \cap x_2 \cup \bar{x}_1 \cap \bar{x}_2) \cdot b \cup (\bar{x}_1 \cap x_2) \cdot c,$$

$$x_0 = \bar{x}_1 \cap \bar{x}_2.$$

By writing the letters a, b, c on the other side of the functions we obtain the left-language equations

$$x_1 = a \cdot (x_1 \cap \bar{x}_2 \cup \bar{x}_1 \cap x_2) \cup b \cdot (x_1) \cup c \cdot (x_1 \cup \bar{x}_2),$$

$$x_2 = a \cdot (\bar{x}_2) \cup b \cdot (x_1 \cap x_2 \cup \bar{x}_1 \cap \bar{x}_2) \cup c \cdot (\bar{x}_1 \cup x_2),$$

$$x_0 = \bar{x}_1 \cap \bar{x}_2.$$

This system has the unique solution $(L(\boldsymbol{A}))^\rho$ and yields immediately the desired boolean automaton $\boldsymbol{B} = (\{a, b, c\}, \{x_1, x_2\}, \mu, \bar{x}_1 \cap \bar{x}_2, \emptyset)$, μ given by

	a	b	c
x_1	$x_1 \cap \bar{x}_2 \cup \bar{x}_1 \cap x_2$	x_1	$x_1 \cup \bar{x}_2$
x_2	$\bar{x}_2$	$x_1 \cap x_2 \cup \bar{x}_1 \cap \bar{x}_2$	$\bar{x}_1 \cup x_2$

Therefore $L(\boldsymbol{B}) = (L(\boldsymbol{A}))^\rho$. This can also be verified directly by constructing A^ρ and $\boldsymbol{A}_{\boldsymbol{B}}$, and comparing the two automata.

It is known that the reverse of any language accepted by an n-state boolean automaton can be accepted by a deterministic automaton with 2^n states (see [4]). Thus, if we define a language R to be of boolean complexity n if the smallest boolean automaton accepting L has n states, this observation together with Theorem 1 yields:

Corollary 1. *The languages of boolean complexity at most n are exactly the reverses of the languages of deterministic complexity at most* 2^n.

Theorem 1 has an immediate consequence for languages over a one-letter alphabet.

Corollary 2. *If R is a regular language of deterministic complexity m, and R is over a*

one-letter alphabet, then there exists a boolean automaton $\boldsymbol{B}$ *with* $\lceil \log_2 m \rceil$ *states which accepts* R, $L(\boldsymbol{B}) = R$.

The proof is contained in the observation that $L = L^\rho$ if L is any language over a one-letter alphabet.

The reader may recall that the situation was different in the case of nondeterministic automata. This might prompt one to speculate that we can always achieve a logarithmic reduction. This, however, turns out to be wrong as the following example shows.

Consider the reduced automaton $\boldsymbol{A} = (\{0, 1, 2\}, \{q_0, q_1\}, \tau, q_0, \{q_0\})$, τ given by

	0	1
q_0	q_1	q_1
q_1	q_0	q_1

If the above conjecture were correct there would exist a boolean automaton with one state accepting $L(\boldsymbol{A})$. This is not true as one can verify directly. Thus there is no boolean automaton for this language with fewer states than two, the number of states of the reduced automaton.

It is an open problem whether there is a regular language of complexity n for all $n \geqslant 3$ such that the smallest boolean automaton has n states.

We now direct our attention to the question whether there are regular languages which can be optimally represented by boolean automata, i.e. we ask whether there are boolean automata with n states, $n \geqslant 1$, such that the corresponding languages are of maximal (deterministic) complexity, namely $2^{(2^n)}$. This question can be answered affirmatively.

Proposition 1. *For every* $m \geqslant 1$ *there exists a deterministic finite automaton* $\boldsymbol{A}_m$ *with* m *states such that* $\boldsymbol{A}^\rho$ *has* 2^m *states.*

Proof. Let $m \geqslant 2$. Consider the following automaton $\boldsymbol{A}_m = (\{a, b, c\}, \{0, \ldots, m-1\}, \tau_m, 0, \{0\})$, τ_m being defined by

$$\tau_m(i, a) = (i+1) \bmod m,$$

$$\tau_m(i, b) = \begin{cases} 1, & i = 0, \\ 0, & i = 1, \\ i, & i = 2, \ldots, m-1, \end{cases}$$

$$\tau_m(i, c) = \begin{cases} m-1, & i = 0, \\ i, & i = 1, \ldots, m-1. \end{cases}$$

For instance the automaton $\boldsymbol{A}$ in the above example is $\boldsymbol{A}_4$. – We claim that $\boldsymbol{A}_m^\rho$ has 2^m states. The proof relies on the following observations. Columns $\tau_m(\cdot, a)$ and $\tau_m(\cdot, b)$, considered as permutations of the set $\{0, \ldots, m-1\}$, generate the symmetric

group S_m of all permutations of the set $\{0, \ldots, m-1\}$. The column $\tau_m(\cdot, c)$ will be used in the construction of $\boldsymbol{A}_m^\rho$ to obtain a set with $j+1$ elements from one with j elements, for $j = 1, \ldots, m-1$. It also induces the empty set. Hence all 2^m subsets $\{0, \ldots, m-1\}$ occur as states of $\boldsymbol{A}_m^\rho$; note that $\boldsymbol{A}_m$ is connected.

It should be clear that this proof is very similar to the proof that there exists a nondeterministic automaton with m states such that the derived deterministic automaton has 2^m states. However, in our case we have the added advantage that $\boldsymbol{A}_m^\rho$ is always reduced.

Combining Theorem 1 and Proposition 1 yields

Theorem 2. *For any $n \geq 1$ there exists a boolean automaton $\boldsymbol{B}_n$ with n states such that the reduced automaton accepting $L(\boldsymbol{B}_n)$ is $\boldsymbol{A}_{\boldsymbol{B}_n}$ and has $2^{(2^n)}$ states.*

Proof. Let $m = 2^n$ in Proposition 1; then $\boldsymbol{A}_m^\rho$ is reduced and has $2^{(2^n)}$ states. Now apply Theorem 1 to $\boldsymbol{A}_m$. The resulting boolean automaton $\boldsymbol{B}_n$ has $n = \lceil \log_2 m \rceil$ states and defines $L(\boldsymbol{A}_m^\rho)$.

Therefore $\boldsymbol{A}_{\boldsymbol{B}_n}$ has $2^{(2^n)}$ states.

Clearly, all constructions are effective. Furthermore the complexity of the construction is substantially 'smaller' than the deterministic automaton we define by specifying a regular language.

Note that the fact that $\boldsymbol{A}_{\boldsymbol{B}_n}$ is reduced is a trivial by-product. This is in marked contrast to the problem for nondeterministic automata where it is quite difficult to show that there is an m-state automaton such that the subset construction yields a deterministic automaton with 2^m states which is *reduced.*

Theorem 2 was first obtained by Kozen [3]; however his method is quite different from ours. Furthermore, it requires an alphabet with three letters, leaving open the question whether fewer letters will suffice.

In view of Corollary 2, Theorem 2 cannot hold for languages over a one-letter alphabet. This can also be seen directly if one considers the two constant boolean functions 0 and 1. However, it does hold for a two-letter alphabet. All we have to show is an analogue of Proposition 2 over an alphabet with two letters.

Proposition 2. *Let* $\hat{\boldsymbol{A}}_m = (\{0, 1\}, \{0, \ldots, m-1\}, \hat{\tau}_m, 1, \hat{F}_m)$ *for* $m \geq 2$ *with* $\hat{F}_m = \{0 \leq i \leq m-1 \mid i \text{ even}\}$ and

$$\hat{\tau}_m(i, 0) = (i+1) \bmod m, \qquad \hat{\tau}_m(i, 1) = \begin{cases} i, & 0 \leq i \leq m-3, \\ m-1, & i = m-2, m-1. \end{cases}$$

Then $(\hat{\boldsymbol{A}}_m)^\rho$ *has* 2^m *states.*

Proof. Clearly $(\hat{\boldsymbol{A}}_m)^\rho = (\{0, 1\}, \{0, \ldots, m-1\}, \delta_m, \hat{F}_m, \{1\})$ and δ_m as follows:

$$\delta_m(i,0) = (i-1) \bmod m, \qquad \delta_m(i,1) = \begin{cases} \{i\}, & 0 \leqslant i \leqslant m-3, \\ 0, & i = m-2, \\ \{m-2, m-1\}, & i = m-1. \end{cases}$$

We have to show that any subset S of $\{0, \ldots, m-1\}$ can be obtained from $\hat{F}_m$ using the actions of $\delta_m(\cdot, 0)$ and $\delta_m(\cdot, 1)$. For convenience let $S = (i_1, \ldots, i_k)$ where $0 \leqslant i_1 < \cdots < i_k \leqslant m-1$. We first observe:

(a) $i_s \in S$ can be erased from S (we can construct $S - \{i_s\}$) if $i_s + 1$ is *not in* S, and

(b) $l \notin S$ can be inserted into S (we can construct $S \cup \{l\}$) if $l+1$ is *in* S.

To verify this, one first uses the cyclic permutation p, enacted by $\delta_m(\cdot, 0)$, x times to get i_s (l) into position $m-2$, then uses $\delta_m(\cdot, 1)$ once, and then applies $(m-x)$ times p again.

The second crucial idea is the following construction of S^{mx}: S^{mx} is the sequence formed out of the maxima of all contiguous subsequences of S. For example, if $S = (1, 2, 3, 7, 8, 10, 12, 15, 16, 17, 18, 19)$ then $S^{mx} = (3, 8, 10, 12, 19)$. Given S^{mx} it is a trivial observation how to construct S. Simply apply (b) to fill in all missing states. Note that S^{mx} can never have more than $\lfloor \frac{1}{2}(m+1) \rfloor$ elements. This is exactly the number of elements in $\hat{F}_m$, the set of starting states of $(\hat{\boldsymbol{A}}_m)^p$. Now is is easy to see that all possible S^{mx} can be obtained from $\hat{F}_m$.

This gives rise to the following corollaries.

Corollary 3. *For every $n \geqslant 1$ there exists a boolean automaton $\boldsymbol{B}_n$ with n states over a two-letter alphabet such that $L(\boldsymbol{B}_n)$ has complexity $2^{(2^n)}$.*

Proof. If $n \geqslant 2$, the result follows from Proposition 2 and Theorem 1.

For $n = 1$ we define the boolean automata $\boldsymbol{B}_n$ as follows:

$$\boldsymbol{B}_1 = (\{0, 1\}, \{q\}, \tau_1, q, \emptyset)$$

with τ_1 given by $\tau_1(q, 0) = \bar{q}$, $\tau_1(q, 1) = 0$.

It can be easily verified that $L(\boldsymbol{B}_1)$ is of deterministic complexity 4.

Corollary 4. *For every $m \geqslant 1$ there exists a nondeterministic automaton $\boldsymbol{N}_m$ with m states over a two-letter alphabet such that $L(\boldsymbol{N}_m)$ is of complexity 2^m.*

Proof. Consider $\hat{\boldsymbol{A}}_m$ defined in Proposition 2 and define $\boldsymbol{N}_m$ as follows:

$$\boldsymbol{N}_m = (\{0, 1\}, \{0, \ldots, m-1\}, (\hat{\tau}_m)^{-1}, \hat{F}_m, \{0\}).$$

Note that $(\hat{\tau}_m)^{-1}$, the inverse function of $\hat{\tau}_m$, is indeed the transition function of a nondeterministic automaton. It is easily verified that the deterministic automaton obtained by the subset construction from $\boldsymbol{N}_m$ is precisely $(\hat{\boldsymbol{A}}_m)^p$, and hence it has 2^m states and is reduced. For $m = 1$, the claim follows trivially.

References

[1] J.A. Brzozowski, Canonical regular expressions and minimal state graph for definite events, in: *Mathematical Theory of Automata*, Symposia Series **12** (Polytechnic Institute of Brooklyn, Brooklyn, 1963) 529–561.

[2] J.A. Brzozowski and E. Leiss, On equations for regular languages, finite automata, and sequential networks, *Theor. Comput. Sci.* **10** (1980) 19–35.

[3] D. Kozen, On parallelism in turing machines, *Proc. 17th Annual Symposium on Foundations of Computer Science* (1976) 89–97.

[4] D. Kozen, private communication, April 1980.