A LINEAR ALGORITHM FOR TESTING EQUIVALENCE OF FINITE AUTOMATA

J. E. Hopcroft
Cornell University
Ithaca, New York

R. M. Karp
University of California
Berkeley, California

TR 71 - 114

December 1971

A LINEAR ALGORITHM FOR TESTING EQUIVALENCE OF FINITE AUTOMATA

J. E. Hopcroft
Cornell University
Ithaca, New York

R. M. Karp
University of California
Berkeley, California

ABSTRACT

An algorithm is given for determining if two finite automata with start states are equivalent. The asymptotic running time of the algorithm is bounded by a constant times the product of the number of states of the larger automaton with the size of the input alphabet.

A LINEAR ALGORITHM FOR TESTING* EQUIVALENCE OF FINITE AUTOMATA

J. E. Hopcroft
Cornell University
Ithaca, New York

R. M. Karp
University of California
Berkeley, California

I. Introduction

The algorithms for testing equivalence of finite automata given in most texts [1,5] have asymptotic growth rates proportional to the square of the number of states. A recent algorithm [2] minimizes the number of states in a finite automaton in $0(n \log n)$ steps where n is the number of states. Clearly the minimization algorithm can be used to test the equivalence of two finite automata by treating them as a single automaton, minimizing the number of states, and seeing if their respective start states are equivalent. However, if the finite automata have start states and one wishes solely to test equivalence, then the algorithm presented here can be used and the running time is bounded by a constant times the product of the number of states in the larger automaton and the size of the input alphabet.

The algorithm makes use of a linear list merging algorithm described elsewhere [3]. The linear list merging algorithm starts with n sets, each set consisting of a single integer between 1 and n. The set containing the integer i is given the name i. The list merging algorithm executes two types of instructions, a merge instruction and a find instruction. The execution of an instruction MERGE(i,j,k) causes the set named i and the set named j to be combined into a single set named k.

*This research was supported by the National Science Foundation

The execution of an instruction FIND(i) determines the name of the set currently containing i. The important property of the algorithm is that the time necessary to execute any sequence of merge and find instructions, whose length does not exceed a constant times n, is bounded by a constant times n.

II. Notation

A finite automaton M is a 5-tuple $M = (S, I, \delta, q_0, F)$ where S and I are finite sets of states and input symbols, respectively; δ is a function mapping $S \times I$ into S, q_0 in S is the start state and $F \subseteq S$ is the set of final states. The function δ is extended from $S \times I$ into S to $S \times I^*$ into S in the obvious manner [4] where I^* is the set of all finite length strings of symbols from I. Let

$M_1 = (S_1, I, \delta_1, q_0, F_1)$ and $M_2 = (S_2, I, \delta_2, p_0, F_2)$

be finite automata. To simplify notation we assume S_1 and S_2 are disjoint and define $\delta(q,a) = \delta_i(q,a)$ for q in S_i, $1 \leq i \leq 2$. An equivalence relation $\equiv$ over $S_1 \cup S_2$ is called a right-invariant equivalence relation if, for all q and p in $S_1 \cup S_2$, and all a in I, $q \equiv p$ implies $\delta(q,a) \equiv \delta(p,a)$. States q and p in $S_1 \cup S_2$ are said to be equivalent if for all x in I^*, $\delta(q,x) \in F_1 \cup F_2$ if and only if $\delta(p,x) \in F_1 \cup F_2$. M_1 and M_2 are said to be equivalent if q_0 is equivalent to p_0.

III. Algorithm for testing the equivalence of finite automata.

The algorithm for testing the equivalence of finite automata makes use of the following observations. If M_1 and M_2 are equivalent, then q_0 and p_0 must be equivalent. If states q and p are equivalent then for each a in I, $\delta(q,a)$ and $\delta(p,a)$ must be equivalent. The algorithm starts by setting up a set for each state, and then merging two sets whenever it is discovered that a state in one set must be equivalent to a state in the other if M_1 is to be equivalent to M_2. Whenever two sets are combined, a state from each set is selected and for each a in I, the sets containing the pair of successor states are combined. When the point is reached where every pair of states in the same set has its successor pair for each a in I in a single set, the process is stopped. M_1 and M_2 are equivalent if and only if at this point no set contains both a final and a non final state.

Step 1: Initialize the linear list merging algorithm with $n = |S_1| + |S_2|$. That is, set up n sets each containing a single element corresponding to a state in $S_1 \cup S_2$. The set containing the state q is assigned the name q.

Step 2: Execute the instruction MERGE(q_0, p_0, p_0) and place the pair (q_0, p_0) on a pushdown store.

Step 3: While the pushdown store is nonempty do the following.

(a) Pop the top pair (q_1,q_2) from the pushdown store.

(b) For each a in I

(i) execute instructions FIND($\delta(q_1,a)$) and FIND($\delta(q_2,a)$).

(ii) Let r_1 and r_2 be the names of the lists containing $\delta(q_1,a)$ and $\delta(q_2,a)$ respectively. If r_1 is not equal to r_2, then execute the instruction MERGE(r_1,r_2,r_2) and place the pair (r_1,r_2) on the pushdown store.

Step 4: Scan the states on each list. The two finite automata are equivalent if and only if no list contains both a final and a non final state.

IV Analysis of the algorithm

We assume that the algorithm is executed on a random access computer.

Theorem 1: The execution time of the algorithm for testing equivalence of finite automata is bounded by a constant times the product of the number of input symbols with the sum of the number of states of each of the automata.

Proof: Steps 1, 2 and 4 are executed in an amount of time bounded by a constant times n. Let m be the cardinality of the set I. The time to execute Step 3 is bounded by a constant times m times the number of pairs popped from the pushdown store. It remains to show that the number of pairs popped from the pushdown store is bounded by n. Each time a pair is placed on the pushdown store, two sets are merged and thus the total number of sets is decreased by one. Since initially there are only n sets, at most $n-1$ pairs are placed on the pushdown store.

For the next lemma we need the following definition. At a given step in the execution of the algorithm, a sequence of states $q_1, q_2, \ldots, q_r$ is said to be a connecting sequence if for $1 \leq i < r$ either

(1) for all $a \in I$ $\delta(q_i,a)$ and $\delta(q_{i+1},a)$ are on the same list, or

(2) the pair (q_i, q_{i+1}) is on the pushdown store.

States q and p are said to be joined by the connecting sequence $q_1, q_2, \ldots q_r$ if $q = q_1$ and $p = q_r$.

Lemma 1: Let E be an equivalence relation on $S_1 \cup S_2$ defined by $q \, E \, p$ if and only if q and p appear on the same list at the termination of the algorithm. Then E is the coarsest right invariant equivalence relation which identifies q_0 and p_0.

Proof: Clearly E is an equivalence relation and identifies q_0 and p_0. Two lists are merged at Step 3bii only if there exist p_1 and p_2, already on the same list, and an a in I such that $\delta_1(p_1,a)$ and $\delta_2(p_2,a)$ are on different lists. Hence the algorithm does not make too many identifications.

That E is right invariant can be proved by induction as follows. Induction hypothesis: Immediately prior to the kth execution of the body of the while statement in Step 3 if states q and p are on the same list then q and p are joined by a connecting sequence.

Clearly the induction hypothesis is true the first time the body of the while statement is executed since q_0 and p_0 are the only states which are on the same list and the pair (q_0,p_0) is on the pushdown store. Thus q_0,p_0 is a connecting sequence joining q_0 and p_0.

If states p and q are joined prior to the kth execution of the body of the while statement, then they are joined after the kth execution. Whenever two lists are merged during the kth execution, a state on the first list is joined to a state on the second list. Assume p and q are on the same list after the kth execution. Consider two cases.

Case 1: States p and q were on the same list prior to the kth execution in which case they were joined and hence remain joined.

Case 2: States p and q end up on the same list as a result of a sequence of merges during the kth execution. In this case several lists have been merged into one list. Each time a pair of lists were merged a state in one list was joined to a state in the other. Since the join relation is reflexive, transitive and symmetric, every pair of elements on the new list are joined. Thus after the <u>kth</u> execution states p and q are joined.

<u>Theorem 2</u>: The algorithm for testing the equivalence of finite automata is correct.

<u>Proof</u>: Combine M_1 and M_2 into a single automaton

$$M_3 = (S_1 \cup S_2, I, \delta, q, F = F_1 \cup F_2) .$$

Let E' be the equivalence relation q E' p if and only if for all x in I, $\delta(q,x)$ is in F if and only if $\delta(p,x)$ is in F. If M_1 is equivalent to M_2, then q_0 E' p_0 and since E' is right invariant, then E' must be a refinement (possibly trivial) of E. Since E' does not identify any final and nonfinal states, E cannot. Therefore, if M_1 and M_2 are equivalent, no list can contain both a final and nonfinal state.

It remains to show that if M_1 is not equivalent to M_2, then some list must contain a final and nonfinal state. Clearly without loss of generality we can assume there exists an x xuch that $\delta_1(q_0,x)$ is in F_1 and $\delta_2(p_0,x)$ is in F_2. Since E is right invariant $\delta_1(q_0,x)$ E $\delta_2(p_0,x)$ and hence $\delta_1(q_0,x)$ and $\delta_2(p_0,x)$ are on the same list. Therefore the list contains both a final and a nonfinal state.

REFERENCES

[1] Harrison, M. A., Introduction to Switching and Automata Theory, McGraw-Hill, New York, 1965.

[2] Hopcroft, J. E., "An N log N Algorithm for Minimizing States in a Finite Automaton," Proceedings of the International Symposium on the Theory of Machines and Computations, Academic Press, New York, 1971 (pp 189-196).

[3] Hopcroft, J. E. and J. D. Ullman, "A Linear List Merging Algorithm," Cornell University, TR 71-111, November, 1971.

[4] Hopcroft, J. E. and J. D. Ullman, Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Massachusetts, 1969.

[5] McCluskey, E. J. Introduction to the Theory of Switching Circuits, McGraw Hill, New York, 1965.