Abstract

This paper presents a finite automata minimization algorithms. Brzozowski's elegant minimization algorithm differs from all other known minimization algorithms, and is derived separately. All of the remaining algorithms depend upon computing an equivalence relation on states. Here, is defined the equivalence relation, the partition that it induces, and its complement. Additionally, some useful properties are derived. It is shown that the equivalence relation is the greatest fixed point of an equation, providing a useful characterization of the required computation. An upperbound on the number of approximation steps is required to compute the fixed point. Algorithms computing the equivalence relation (or the partition, or its complement) are derived systematically in the same framework.

Introduction

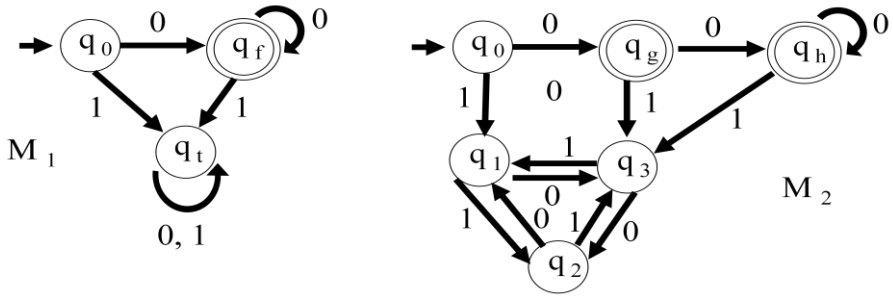
1.Minimizing Finite Automata

DFA minimization is the task of transforming a given deterministic finite automaton (DFA) into an equivalent DFA that has minimum number of states. Here, two DFAs are called equivalent if they recognize the same regular language

1.1Minimum DFA

For each regular language that can be accepted by a DFA, there exists a **minimal automaton**, a DFA with a minimum number of states and this DFA is unique (except that states can be given different names.) The minimal DFA ensures minimal computational cost for tasks such as pattern matching. [1]
For a given regular language L, it is possible to construct infinitely many finite state automata to accept L. The reason is that based on a given DFA M_1, there would be a loop in the transition diagram. So we can construct a new DFA M_2 to have more states and $L(M_1) = L(M_2) = L$. And from DFA M_2 , we can construct a DFA M_3 with even more states and $L(M_3) = L$.
Consider example 1, given a DFA M_1, we can construct a DFA M_2 to have more states and $L(M_1) = L(M_2) = L$.

Example 1: The following two DFA's M_1 and M_2 are equivalent. Both machines accept the set $\{0^n | n>0\}$ over the alphabet $\Sigma=\{0, 1\}$.



The states q_1, q_2, and q_3 of the machine M_2 are equivalent, we can combine these 3 states into one state as q_t in the machine M_1.

The states q_g and q_h of the machine M_2 are equivalent, we can combine these 2 states into one state as q_f in the machine M_1. [7]

There are two classes of states that can be removed/merged from the original DFA without affecting the language it accepts to minimize it.

- **Unreachable states** are those states that are not reachable from the initial state of the DFA, for any input string.

- **Non-distinguishable states** are those that cannot be distinguished from one another for any input string.

DFA minimization is usually done in three steps, corresponding to the removal/merger of the relevant states. Since the elimination of non-distinguishable states is computationally the most expensive one, it is usually done as the last step.

a.) Unreachable states

The state p of DFA $M=(Q, \Sigma, \delta, q_0, F)$ is unreachable if no such string w in $\sum^*$ exists for which $p=\delta(q_0, w)$.
Unreachable states can be removed from the DFA without affecting the language that it accepts.

b.)Non-distinguishable states

2.Algorithms for minimizing Finite Automata

2.1 Hopcroft's algorithm

One algorithm for merging the non-distinguishable states of a DFA, due to Hopcroft (1971), is based on partition refinement, partitioning the DFA states into groups by their behavior. These groups represent equivalence classes of the Myhill–Nerode equivalence relation, whereby every two states of the same partition are equivalent if they have the same behavior for all the input sequences. That is, for every two states p_1 and p_2 that belong to the same equivalence class within the partition P, it will be the case that for every input word w, if one follows the transitions determined by w from the two states p_1 and p_2 one will either be led to accepting states in both cases or be led to rejecting states in both cases; it should not be possible for w to take p_1 to an accepting state and p_2 to a rejecting state or vice versa.
Every pair of states that are equivalent according to the Myhill–Nerode relation belong to the same set in the partition, but pairs that are inequivalent might also belong to the same set. It gradually refines the partition into a larger number of smaller sets, at each step splitting sets of states into pairs of subsets that are necessarily inequivalent. The initial partition is a separation of the states into two subsets of states that clearly do not have the same behavior as each other: the accepting states and the rejecting states. The algorithm then repeatedly chooses a set A from the current partition and an input symbol c, and splits each of the sets of the partition into two (possibly empty) subsets: the subset of states that lead to A on input symbol c, and the subset of states that do not lead to A. Since A is already known to have different behavior than the other sets of the partition, the subsets that lead to A also have different behavior than the subsets that do not lead to A. When no more splits of this type can be found, the algorithm terminates.
Once Hopcroft's algorithm has been used to group the states of the input DFA into equivalence classes, the minimum DFA can be constructed by forming one state for each equivalence class. If S is a set of states in P, s is a state in S, and c is an input character, then the transition in the minimum DFA from the state for S, on input c, goes to the set containing the state that the input automaton would go to from state s on input c. The initial state of the minimum DFA is the one containing the initial state of the input DFA, and the accepting states of the minimum DFA are the ones whose members are accepting states of the input DFA.
The worst case running time of this algorithm is $O(ns \log n)$, where n is the number of states and s is the size of the alphabet. This bound follows from the fact that, for each of the ns transitions of the automaton, the sets drawn from Q that contain the target state of the transition have sizes that decrease relative to each other by a factor of two or more, so each transition participates in $O(\log n)$ of the splitting steps in the algorithm. The partition refinement data structure allows each splitting step to be performed in time proportional to the number of transitions that participate in it. This remains the most efficient algorithm known for solving the problem, and for certain distributions of inputs its average-case complexity is even better, $O(n \log \log n)$.

2.2 Moore's algorithm

Moore's algorithm for DFA minimization is due to Edward F. Moore (1956). Like Hopcroft's algorithm, it maintains a partition that starts off separating the accepting from the rejecting states, and repeatedly refines the partition until no more refinements can be made. At each step, it replaces the current partition with the coarsest common refinement of s + 1 partitions, one of which is the current one and the others are the pre-images of the current partition under the transition functions for each of the input symbols. The algorithm terminates when this replacement does not change the current partition. Its worst-case time complexity is $O(n^2s)$: each step of the algorithm may be performed in time O(ns) using a variant of radix sort to reorder the states so that states in the same set of the new partition are consecutive in the ordering, and there are at most n steps since each one but the last increases the number of sets in the partition. The instances of the DFA minimization problem that cause the worst-case behaviour are the same as for Hopcroft's algorithm. The number of steps that the algorithm performs can be much smaller than n, so on average (for constant s) its performance is O(n log n) depending on the random distribution on automata chosen to model the algorithm's average-case behavior.

2.3 Brzozowski's algorithm

As Brzozowski (1963) observed, reversing the edges of a DFA produces a non-deterministic finite automaton (NFA) for the reversal of the original language, and converting this NFA to a DFA using the standard powerset construction (constructing only the reachable states of the converted DFA) leads to a minimal DFA for the same reversed language. Repeating this reversal operation a second time produces a minimal DFA for the original language. The worst-case complexity of Brzozowski's algorithm is exponential, as there are regular languages for which the minimal DFA of the reversal is exponentially larger than the minimal DFA of the language, but it frequently performs better than this worst case would suggest.

NFA Minimization

While the above procedures work for DFAs, the method of partitioning does not work for non-deterministic finite automata (NFAs). While an exhaustive search may minimize an NFA, finding a polynomial-time algorithm to minimize NFAs is impossible, unless the unsolved conjecture P=PSPACE in computational complexity theory is true. This conjecture is widely believed to be false. [1]

---

4. Example of minimizing finite automata

Consider the finite automaton shown in figure 1 which accepts the regular set denoted by the regular expression $(aa + b)^*ab(bb)^*$. Accepting states are s2 and s7 while rejecting states are s0, s1, s3, s4, s5, s6.

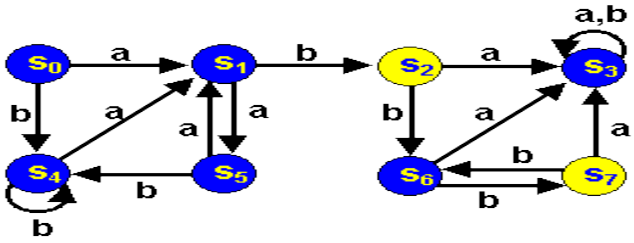


Figure 1 - Recognizer for (aa + b)*ab(bb)*

Closer examination reveals that states s_2 and s_7 are really the same since they are both accepting states and both go to s_6 under the input b and both go to s_3 under an a. So, why not merge them and form a smaller machine? In the same manner, we could argue for merging states s_0 and s_5. Merging states like this should produce a smaller automaton that accomplishes exactly the same task as our original one. From these observations, it seems that the key to making finite automata smaller is to recognize and merge equivalent states. To do this, we must agree upon the definition of equivalent states. Here is one formulation of what Moore defined as indistinguishable states.

Definition. Two states in a finite automaton M are **equivalent** if and only if for every string x, if M is started in either state with x as input, it either accepts in both cases or rejects in both cases.

Another way to say this is that the machine does the same thing when started in either state.

For a deterministic finite automaton M, the minimum number of states in any equivalent deterministic finite automaton is the same as the number of equivalence classes of M's states.

Now , we know that if we can find the equivalence classes (or groups of equivalent states) for an automaton, then we can use these as the states of the smallest equivalent machine. The machine shown in figure 1 will be used as an example for the intuitive discussion that follows.

Let us first divide the machine's states into two groups: accepting and rejecting states. These groups are: A = {s_2, s_7} and B = {s_0, s_1, s_3, s_4, s_5, s_6}. Note that these are equivalent under the empty string as input.

Then, let us find out if the states in these groups go to the same group under inputs a and b. As we noted at the beginning of this discussion, the states of group A both go to states in group B under both inputs. Things are different for the states of group B. The following table shows the result of applying the inputs to these states. (For example, the input a leads from s_1 to s_5 in group B and input b leads to to s_2 in group A.)

in state:	s_0	s_1	s_3	s_4	s_5	s_6
a leads to:	B	B	B	B	B	B
b leads to:	B	A	B	B	B	A

Looking at the table we find that the input b helps us distinguish between two of the states (s1 and s6) and the rest of the states in the group since it leads to group A for these two instead of group B. Thus the states in the set {s_0, s_3, s_4, s_5} cannot be equivalent to those in the set {s_1, s_6} and we must partition B into two groups. Now we have the groups:

A = {s_2, s_7}, B = { s_0, s_3, s_4, s_5}, C = { s_1, s_6}

and the next examination of where the inputs lead shows us that s_3 is not equivalent to the rest of group B. We must partition again.

Continuing this process until we cannot distinguish between the states in any group by employing our input tests, we end up with the groups:

A = {s_2, s_7}, B = {s_0, s_4, s_5}, C = {s_1}, D = {s_3}, E = { s_6}.

In view of the above theoretical definitions and results, it is easy to argue that all of the states in each group are equivalent because they all go to the same groups under the inputs a and b. Thus in the sense of Moore the states in each group are truly indistinguishable. We also can claim that due to the corollary to the Myhill-Nerode theorem, any automaton that accepts $(aa + b)^*ab(bb)^*$ must have at least five states.

Building the minimum state finite automaton is now rather straightforward, merely use the equivalence classes (our groups) as states and provide the proper transitions. This gives us the finite automaton pictured in figure 2.

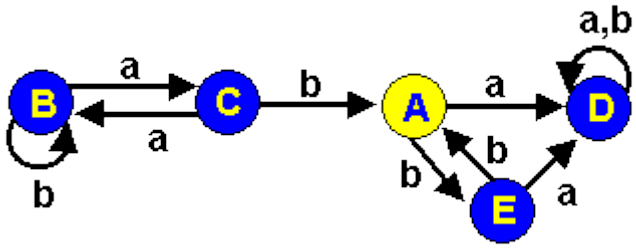
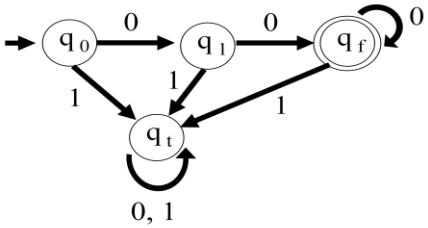


Figure 2 - A Minimal Automaton

Two states p and q of a DFA M are **equivalent**, if for all strings ω in Σ^* both the states δ(p, ω) and δ(q, ω) are in F or both are not in F. Two states p and q are **distinguishable**, if they are not equivalent. [2]

Example : Given a DFA M as follows, Find the equivalence classes of R_M.



Solution :

The equivalence classes of R_M are :

$[q_0] = \{\varepsilon\}, \quad [q_1] = \{0\}, \quad [q_f] = \{ 0^n \mid n > 1 \},$

$[q_t] = \Sigma^* \setminus \{ 0^n \mid n > 0 \}.$ [7]

Literature Survey

A Taxonomy of Finite Automata Minimization Algorithms (1993) by Bruce Watson

This paper presents a taxonomy of finite automata minimization algorithms. Brzozowski's elegant minimization algorithm differs from all other known minimization algorithms, and is derived separately. All of the remaining algorithms depend upon computing an equivalence relation on states. We define the equivalence relation, the partition that it induces, and its complement. Additionally, some useful properties are derived. It is shown that the equivalence relation is the greatest fixed point of an equation, providing a useful characterization of the required computation. We derive an upperbound on the number of approximation steps required to compute the fixed point. Algorithms computing the equivalence relation (or the partition, or its complement) are derived systematically in the same framework. The algorithms include Hopcroft's, several algorithms from text-books (including Hopcroft and Ullman's [HU79], Wood's [Wood87], and Aho, Sethi, and Ullman's [ASU86]), and several new algorithm.[3]

Cycle-aware minimization of acyclic deterministic finite-state automata
By Johannes Bubenzer
University of Potsdam, Department of Linguistics, Karl-Liebknecht-Strasse 24-25, 14476 Potsdam, Germany

In this paper a linear-time algorithm for the minimization of acyclic deterministic finite-state automata is presented. The algorithm runs significantly faster than previous algorithms for the same task. This is shown by a comparison of the running times of both algorithms. Additionally, a variation of the new algorithm is presented which handles cyclic automata as input. The new cycle-aware algorithm minimizes acyclic automata in the desired way. In case of cyclic input, the algorithm minimizes all acyclic suffixes of the input automaton.[4]

Edge-minimization of non-deterministic finite automata
By B. F. Melnikov, A. A. Melnikova

In this paper we consider non-deterministic finite Rabin-Scott's automata. We use a special structure to describe all the possible edges of non-deterministic finite automaton defining the given regular language. Such structure can be used for solving various problems of finite automata theory. One of these problems is edge-minimization of non-deterministic automata. As we have not touched this problem before, we obtain here two versions of the algorithm for solving this problem to continue previous series of articles.[5]

Minimizing Finite Automata Is Computationally Hard
By Andreas Malcher

It is known that deterministic finite automata (DFAs) can be algorithmically minimized, i.e., a DFA Mcan be converted to an equivalent DFA M' which has a minimal number of states. The minimization can be done efficiently [6]. On the other hand, it is known that unambiguous finite automata (UFAs) and nondeterministic finite automata (NFAs) can be algorithmically minimized too, but their minimization problems turn out to be **NP**-complete and **PSPACE**-complete, respectively [8]. In this paper, the time complexity of the minimization problem for two restricted types of finite automata is investigated. These automata are nearly deterministic, since they only allow a small amount of non-determinism to be used. The main result is that the minimization problems for these models are computationally hard, namely **NP**-complete. Hence, even the slightest extension of the deterministic model towards a nondeterministic one, e.g., allowing at most one nondeterministic move in every accepting computation or allowing two initial states instead of one, results in computationally intractable minimization problems [6]

References:

[1] http://en.wikipedia.org/wiki/DFA_minimization
[2] http://www.cs.engr.uky.edu/~lewis/essays/compilers/min-fa.html
[3] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.9852
[4] http://www.sciencedirect.com/science/article/pii/S0166218X13003442
[5] http://link.springer.com/article/10.1007%2FBF02941980
[6] http://link.springer.com/chapter/10.1007%2F3-540-45007-6_31
[7]https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0CDQQFjAB&url=http%3A%2F%2Fnknucc.nknu.edu.tw%2F~jwu%2Fautomata%2Fauto03s1.ppt&ei=mGqHUtLoFsTtrQfw-YD4Bg&usg=AFQjCNH7WiGaBZ98V4ZFe147CuyM1d-lxQ&sig2=hoAWc6cqPB69ExAQDx2kXg