Notes on State Minimization

(Revised Feb. 11, 2004)

In these notes we present an efficient algorithm to convert a given DFA into a DFA for the same language and with a minimum number of states.

Let L be a language over an alphabet Σ. We have seen in Handout 1 the definition of distinguishable strings with respect to L. We say that two strings x and y are **indistinguishable**, and we write it $x \approx_L y$ if they are not distinguishable. That is, $x \approx_L y$ means that, for every string z, the string xz belongs to L if and only if the string yz does. By definition, $x \approx_L y$ if and only if $y \approx_L x$, and we always have $x \approx_L x$. It is also easy to see that if $x \approx_L y$ and $y \approx_L w$ then we must have $x \approx_L w$. In other words:

Fact 1 *The relation $\approx_L$ is an* **equivalence** *relation over the strings in Σ^*.*

As you may remember from such classes as Math55 or CS70, when you define an equivalence relation over a set you also define a way to partition the set into a collection of subsets, called *equivalence class*. An equivalence class in Σ^* with respect to $\approx_L$ is a set of strings that are all indistinguishable from one another, and that are all distinguishable from all the others not in the set. We denote by $[x]$ the equivalence class that contains the string x.

A fancy way of stating the main result from Handout 1 is to say that every DFA for L must have at least as many states as the number of equivalence class in Σ^* with respect to $\approx_L$. Perhaps surprisingly, the converse is also true: there is always a DFA that has *precisely* as many states as the number of equivalence classes.

Theorem 1 (Myhill-Nerode) *Let L be a language over Σ. If Σ^* has infinitely many equivalence classes with respect to $\approx_L$, then L is not regular. Otherwise, L can be decided by a DFA whose number of states is equal to the number of equivalence classes in Σ^* with respect to $\approx_L$.*

PROOF: If there are infinitely many equivalence classes, then it follows from Lemma 2 in Handout 1 that no DFA can decide L, and so L is not regular.

Suppose then that there is a finite number of equivalence class. We define an automaton that has a state for each equivalence class. The start state is the class $[\epsilon]$ and every state of the form $[x]$ for $x \in L$ is a final state.

It remains to describe the transition function. From a state $[x]$, reading the character a, the automaton moves to state $[xa]$. We need to make sure that this definition makes sense. If $x \approx_L x'$, then the state $[x]$ and the state $[x']$ are the same, so we need to verify that the state $[xa]$ and the state $[x'a]$ are also the same. That is, we need to verify that, for every string z, $xaz \in L$ if and only if $x'az \in L$; this is clearly true because x and x' are indistinguishable and so appending the string az makes xaz an element of the language L if and only if it also makes $x'az$ an element of the language.

So the automaton is well defined. Let now $x = x_1 x_2 \cdots x_n$ be an input string for the automaton. The automaton starts in $[\epsilon]$, then moves to state $[x_1]$, and so on, and at the end is in state $[x_1 \cdots x_n]$; this is an accepting state if and only if $x \in L$, and so the automaton works correctly on x. $\Box$

The Myhill-Nerode theorem shows that one can use the distinguishability method to prove optimal lower bounds on the number of states of a DFA for a given language, but it does not give an efficient way to construct an optimal DFA.

We will now see a *polynomial time* algorithm that given a DFA finds an equivalent DFA with a minimum number of states. There is even an $O(n \log n)$ algorithm for this problem, where $n = |Q| \cdot |\Sigma|$ is the size of the input, but we will not see it.

Before describing the algorithm, consider the following definition.

Definition 1 (Equivalence of states) *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We say that two states $p, q \in Q$ are* **equivalent***, and we write it $p \equiv q$, if for every string $x \in \Sigma^*$ the state that M reaches from p given x is accepting if and only if the state that M reaches from q given x is accepting.*

An equivalent way to think of the definition is that if we change M so that p is the start state the language that we recognize is the same as if we change M so that q is the start state.

We can verify that $\equiv$ is an equivalence relation among the set of states Q, and so $\equiv$ partitions Q into a set of equivalence classes.

The intution behind this definition is that if $p \equiv q$, then it is redundant to have two different states p and q. We would then like to compute the set of equivalence classes of Q with respect to $\equiv$ and then construct a new automaton that has only one state for each equivalence class. The problem here is that it is not clear how to compute the relation $\equiv$, since it looks like we need to test infinitely many cases (one for each string $x \in \Sigma^*$) in order to verify that $p \equiv q$. Fortunately, there is a shortcut. To discuss the shortcut, we need one more definition.

Definition 2 *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We say that two states $p, q \in Q$ are* **equivalent for length** *n, and we write it $p \equiv_n q$, if for every string $x \in \Sigma^*$ of length $\leq n$ the state that M reaches from p given x is accepting if and only if the state that M reaches from q given x is accepting.*

In other words, if we change M so that p becomes the start state, the set of strings of length at most n that we accept is precisely the same as if we had made q be the start state.

The equivalence relations $\equiv_n$ can be computed recursively.

- First, note that $p \equiv_0 q$ if and only if p is accepting and q is not (or vice versa). That is, the equivalence classes for $\equiv_0$ are the set F and the set $Q - F$.
- Suppose we have computed $\equiv_{n-1}$. Now we have that $p \equiv_n q$ if and only if
  - $p \equiv_{n-1} q$ and
  - for every $a \in \Sigma$ we have $\delta(p, a) \equiv_{n-1} \delta(q, a)$.

The second part requires some justification, and we prove it as a lemma below.

Lemma 2 *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. For any two states p, q and integer $n \geq 1$, we have that $p \not\equiv_n q$ if and only if $p \not\equiv_{n-1} q$ or there is an $a \in \Sigma$ such that $\delta(p, a) \not\equiv_{n-1} \delta(q, a)$.*

PROOF: If $p \not\equiv_n q$, then there is a string x such that M accepts when starting from p and given x, but it rejects when starting from q and given x. If the length of x is $\leq n-1$, then we have $p \not\equiv_{n-1} q$. Otherwise, let us write $x = ax'$, where a is the first character of x, and let us call $p' = \delta(p, a)$ and $q' = \delta(q, a)$. Then the string x' has length $n - 1$ and it shows that $p' \not\equiv_{n-1} q'$.

For the other direction, if $p \not\equiv_{n-1} q$ then clearly also $p \not\equiv_n q$. Otherwise, if there is an a such that $\delta(p,a) \not\equiv_{n-1} \delta(q,a)$, then let x' be the string of length $\leq n-1$ that shows that $\delta(p,a) \not\equiv_{n-1} \delta(q,a)$. Then the string ax' of length $\leq n$ shows that $p \not\equiv_n q$. $\square$

This gives an $O(|\Sigma| \cdot |Q|)$ time algorithm to compute $\equiv_n$ given $\equiv_{n-1}$, and so an $O(n \cdot |\Sigma| \cdot |Q|)$ time algorithm to compute $\equiv_n$ from scratch.

Now we come to an important observation: If, for some k, we have that the relation $\equiv_k$ is the same as the relation $\equiv_{k+1}$, then also $\equiv_k$ is the same as $\equiv_n$ for all $n > k$. To see that it is true, note that the process that we use to go from $\equiv_k$ to $\equiv_{k+1}$ is independent of k, and so if it does not change the relation when applied once, it will not change the relation if applied an arbitrary number of times.

This shows that the algorithm converges to a fixed partition in $\leq |Q|-1$ steps, because it starts with two equivalence classes, it cannot create more than $|Q|$ equivalence classes, and at each step it must either increase the number of equivalence classes or reach the final partition.

In particular, if we let $k = |Q|-1$, then we have that $\equiv_k$ is the same as $\equiv_n$ for every $n \geq k$, and it can be computed in time at most $O(|Q|^2 \cdot |\Sigma|)$. We can also see that $\equiv_k$ is the same relation as $\equiv$. This is true because if $p \equiv q$ then certainly $p \equiv_k q$. But also, if $p \not\equiv q$, then there is a string x that shows this is the case, and if call n the length of x we have that $p \not\equiv_n q$. If $n \leq k$, then certainly this also means that $p \not\equiv_k q$; if $n > k$ we can still say that $p \not\equiv_k q$ because we have observed above that $\equiv_n$ and $\equiv_k$ are the same for $n > k$.

We can now finally describe our state minimization algorithm. Given $M = (Q, \Sigma, \delta, q_0, F)$.

- Let $k = |Q| - 1$ and compute the equivalence classes of Q with respect to $\equiv_k$. Define a new automaton $M' = (Q', \Sigma, \delta', q'_0, F')$ as follows. There is a state in Q' for each equivalence class. The initial state q'_0 is the equivalence class $[q_0]$. The set F' contain all the equivalence classes that contain final states in F. [1] Define $\delta'([q], a) = [\delta(q,a)]$.

- Remove from Q' all the states that are not reachable from q'_0. Such states can be easily found doing a depth-first search of the graph of the automaton starting from q'_0. The removal of these states does not change the language accepted by automaton because they never occur in a computation. Let $M'' = (Q'', \Sigma, \delta'', q'_0, F'')$ be the new automaton.

The reader should verify that our definition of δ' makes sense and that the automaton M' decides the same language as M.

To see that the algorithm constructs an optimal automaton, let $t = |Q''|$ be number of states of the automaton. Every state $[q]$ of the automaton is reachable from $[q_0]$, so for every state $[q]$ there is at least a string $x_{[q]}$ such that M' reaches $[q]$ when given in input $x_{[q]}$.

Consider now two different states $[p] \neq [q]$. We have $p \not\equiv q$ in M, and so there must be some string y such that starting from p in M we accept when given y, but starting from q we reject (or vice versa). Also, in M'', starting from $[p]$ we accept when given y but starting from $[q]$ we reject. This means that y shows that the strings $x_{[q]}$ and $x_{[p]}$ are distinguishable, and so we have a set of t distinguishable strings, which implies that it is impossible to decide the same language with fewer than t states.

[1] Note that an equivalence class contains either only states in F or only states not in F.