The Myhill-Nerode Theorem and DFA Minimization

Myhill-Nerode Theorem

So far, we have encountered three necessary and sufficient conditions for determining whether a language L is regular:

- There exists a DFA that recognizes L. (Original definition.)
- There exists an NFA that recognizes L. Proof: Any DFA can be converted to an equivalent NFA (trivial), and any NFA can be converted to an equivalent DFA (power set construction).
- There exists a regular expression that describes L. Proof: Any RE can be converted to an equivalent NFA (by induction), and any DFA can be converted to an equivalent RE (GNFA construction).

In this document, we state yet another necessary and sufficient condition for regular languages. It is called the Myhill-Nerode Theorem. (The theorem can be found in pages 90-91 of Sipser, and its proof is on pages 98-99.) First, we need to set up some preliminary definitions.

Definition Let x and y be strings and let L be any language. We say that x and y are *distinguishable by* L if some string z exists whereby exactly one of the strings xz and yz is a member of L; otherwise, for every string z, we have $xz \in L$ whenever $yz \in L$ and we say that x and y are *indistinguishable by* L. If x and y are indistinguishable by L, we write $x \equiv_L y$.

To put it another way, suppose we have a machine that recognizes language L by processing strings left-to-right (not necessarily a DFA or NFA; it can be any type of automaton). Suppose it is given either string x or string y as input, and afterward, it is given a suffix z. This is equivalent to asking the machine if xz and yz are members of L. If strings x and y are indistinguishable by L, then after reading x or reading y, the machine no longer needs to remember whether it read x, or whether it read y, in order to return the correct accept/reject result. The result only depends on the suffix z. The machine only needs to remember that it read either x or y without remembering which one specifically. If x and y *are* distinguishable by L, then the machine *must* rely on its prior memory of whether it read x or y to return the correct result.

Definition Let L be a language and let X be a set of strings. Say that X is *pairwise distinguishable by* L if every two distinct strings in X are distinguishable by L. Define the *index of* L to be the maximum number of elements in any set that is pairwise distinguishable by L. The index of a language may be finite or infinite.

Put another way, the index of a language L quantifies how much memory a machine needs to recognize language L.

Myhill-Nerode Theorem A language is regular iff it has finite index.

Proof: If L is regular, it is recognized by some DFA with k states, and we can use this fact to show that its index is at most k (and therefore finite). Conversely, if L is a language with a (finite) index of k, we can recognize it with a DFA with k states. The details are on pages 98-99 of Sipser. ■

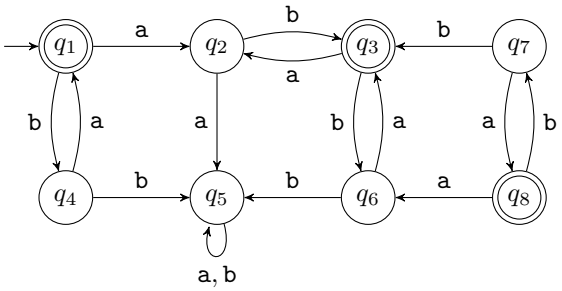
DFA Minimization

The Myhill-Nerode Theorem explicates an intuition you may have already developed when designing finite automata: for regular languages, there is always a point while reading the input string left-to-right when

you can omit details about the symbols read so far and collapse that information into one of a finite number of states. The theorem also implies that for any regular language, there is a unique DFA with a *minimal* number of states (equal to the index of L) that recognizes it. This leads us to an efficient algorithm that reduces the number of states in a DFA to the minimum possible number. It works by identifying equivalent states according to the relation $\equiv_L$.

Minimizing the number of states in a DFA is desirable for a number of reasons. It can reduce the memory requirements of state machines in embedded systems or in software. Conversions from REs and NFAs to DFAs also result in an exponential blowup of the number of DFA states, so we can use DFA minimization to find the optimal DFA. This kind of optimization is not generally possible for other computational models, as we will see later.

We will perform the minimization procedure on the following DFA M, which recognizes the language $L = \{\texttt{ab}, \texttt{ba}\}^*$.

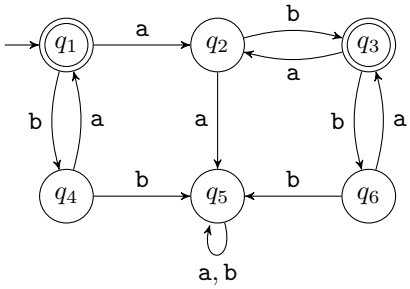


Unreachable States

The first and simplest step in DFA minimization is to remove unreachable states. We can do this using a standard graph traversal algorithm (e.g. breadth-first search or depth-first search), starting at the initial state, marking nodes and removing those that remain unmarked. Here is the pseudocode that would implement this idea:

```
agenda ← {q_0}
visited ← {q_0}
while agenda is not empty do
    remove state q from agenda
    for each a ∈ Σ do
        r ← δ(q, a)
        if r ∉ visited then
            add r to agenda
            add r to visited
        end if
    end for
end while
remove all states not in visited
```

Using this algorithm on M would mark states $\{q_1, q_2, q_3, q_4, q_5, q_6\}$ as visited and remove q_7 and q_8.



Merging Equivalent States

The next phase of DFA minimization involves identifying and merging equivalent states. Informally, two states are equivalent if exactly the same set of strings leads the DFA to acceptance from either state. More formally, states q and r are equivalent if, for any string x that ends in state q and any string y that ends in state r, $x \equiv_L y$. We identify the minimal number of equivalence classes (and therefore the minimal number of states) by iterating through string lengths starting from 0. We first process string of length 0 and see what equivalence classes we can form. Then we increase the string length to 1 and split the existing equivalence classes as necessary. We stop the algorithm when the equivalence classes do not change. Then, all states that belong to the same equivalence class are merged into a single state.

We will define and go through a sequence of equivalence relations $\equiv_0$, $\equiv_1$, $\equiv_2$, ... Here $q \equiv_i r$ means that states q and r remain equivalent after processing a string of length i or less.

For $\equiv_0$ (when we have not read any symbols yet), we know that $q \equiv_0 r$ iff q and r are both accept states, or q and r are both not accept states (because they have the same accept/reject behavior upon reading a string of length 0, which is ε). So there are two equivalence classes for $\equiv_0$: F and $Q - F$. Now we need to know how to compute $\equiv_{i+1}$ from $\equiv_i$. The following lemma shows under what conditions two states within the same equivalence class remain within that class after processing strings which are one character longer.

Lemma 1 For any $q, r \in Q$ and any $i \geq 0$, $q \equiv_{i+1} r$ if and only if

1. $q \equiv_i r$, and
2. for all $a \in \Sigma$, $\delta(q, a) \equiv_i \delta(r, a)$.

When iterating over string lengths 1, 2, ..., whenever there are two states for which Lemma 1 does not hold when we go from length i to length $i + 1$, we will separate those states into two classes. We will need to test each pair of states within each equivalence class. Note that the second condition requires that the states are equivalent for *all* input symbols $a \in \Sigma$. If it fails to hold true on any of them, then we must separate the states into separate classes.

The state minimization algorithm can be summarized as:

set the equivalence classes for $\equiv_0$ to F and $Q - F$
for $i = 0, 1, 2, \ldots$ **do**
  compute the equivalence classes of $\equiv_{i+1}$ from $\equiv_i$ using Lemma 1
  stop if $\equiv_{i+1}$ is the same as $\equiv_i$
end for

For DFA M, we have

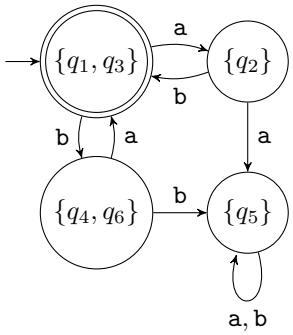
$\equiv_0$: Equivalence classes for $\equiv_0$: $\{q_1, q_3\}$, $\{q_3, q_4, q_5, q_6\}$

$\equiv_1$: Unordered pairs in $\{q_1, q_3\}$:
  $\{q_1, q_3\}$: $\delta(q_1, \mathtt{a}) = q_2 \equiv_0 \delta(q_3, \mathtt{a}) = q_2$ and $\delta(q_1, \mathtt{b}) = q_4 \equiv_0 \delta(q_3, \mathtt{b}) = q_6$ $\rightarrow q_1 \equiv_1 q_3$
  Unordered pairs in $\{q_3, q_4, q_5, q_6\}$:

$\{q_2, q_4\}$: $\delta(q_2, \mathtt{a}) = q_5 \not\equiv_0 \delta(q_4, \mathtt{a}) = q_1$ $\rightarrow q_2 \not\equiv_1 q_4$
$\{q_2, q_5\}$: $\delta(q_2, \mathtt{a}) = q_5 \equiv_0 \delta(q_5, \mathtt{a}) = q_5$ but $\delta(q_2, \mathtt{b}) = q_3 \not\equiv_0 \delta(q_5, \mathtt{b}) = q_5$ $\rightarrow q_2 \not\equiv_1 q_5$
$\{q_2, q_6\}$: $\delta(q_2, \mathtt{a}) = q_5 \not\equiv_0 \delta(q_6, \mathtt{a}) = q_3$ $\rightarrow q_2 \not\equiv_1 q_6$
$\{q_4, q_5\}$: $\delta(q_4, \mathtt{a}) = q_1 \not\equiv_0 \delta(q_5, \mathtt{a}) = q_5$ $\rightarrow q_4 \not\equiv_1 q_5$
$\{q_4, q_6\}$: $\delta(q_4, \mathtt{a}) = q_1 \equiv_0 \delta(q_6, \mathtt{a}) = q_3$ and $\delta(q_4, \mathtt{b}) = q_5 \equiv_0 \delta(q_6, \mathtt{b}) = q_5$ $\rightarrow q_4 \equiv_1 q_6$
$\{q_5, q_6\}$: $q_5 \not\equiv_1 q_4 \equiv_1 q_6$ $\rightarrow q_5 \not\equiv_1 q_6$
Equivalence classes for $\equiv_1$: $\{q_1, q_3\}, \{q_2\}, \{q_4, q_6\}, \{q_5\}$

$\equiv_2$: Unordered pairs in $\{q_1, q_3\}$:
$\{q_1, q_3\}$: $\delta(q_1, \mathtt{a}) = q_2 \equiv_1 \delta(q_3, \mathtt{a}) = q_2$ and $\delta(q_1, \mathtt{b}) = q_4 \equiv_1 \delta(q_3, \mathtt{b}) = q_6$ $\rightarrow q_1 \equiv_2 q_3$
Unordered pairs in $\{q_4, q_6\}$:
$\{q_4, q_6\}$: $\delta(q_4, \mathtt{a}) = q_1 \equiv_1 \delta(q_6, \mathtt{a}) = q_3$ and $\delta(q_4, \mathtt{b}) = q_5 \equiv_1 \delta(q_6, \mathtt{b}) = q_5$ $\rightarrow q_4 \equiv_2 q_6$
Equivalence classes for $\equiv_2$: $\{q_1, q_3\}, \{q_2\}, \{q_4, q_6\}, \{q_5\}$ (No change, so the algorithm terminates.)

We merge states that are in the same equivalence classes, resulting in the following mimimal DFA:



Acknowledgements

This document was based on an earlier document written by Dr. Marina Blanton.