# A minimization algorithm for deterministic finite automata

The following algorithm takes as input a deterministic finite automaton (DFA) and produces an equivalent one (*i.e.* one accepting the same language) with a minimum number of states. The material is taken from [HU79, §3.4].

A DFA is given by a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, where

- $Q$ is a finite set (of *states*)
- $\Sigma$ is a finite set (of *symbols*, forming the *alphabet*)
- $\delta$ is the transition function, assigning to every state $s$ and input symbol $a$ the next state, $\delta(s, a) = s'$ (which may also be written as $s \xrightarrow{a} s'$).
- $q_0$ is a distinguished *start* state
- $F$ is a subset of $Q$ consisting of the *final* or *accepting* states

For the algorithm, we may represent $Q$ by a collection of small integers $Q = \{1, \ldots, n\}$, the alphabet by an enumeration type $\Sigma = [a \ldots z]$ (achieved in Java say, by declaring a collection of integer constants corresponding to the symbols of $\Sigma$) and the transition function by an array of integers doubly indexed by $Q$ and $\Sigma$ (**int**[ ][ ]`Delta`), so that *e.g.* `Delta`$[2][a] = 4$ represents $2 \xrightarrow{a} 4$. The final states $F$ are organised into an array (or list) of integers.

What we need in order to minimize $M$ is to recognize which pairs of states $p, q$ are equivalent (*i.e.*, recognize the same languages: for every string $\sigma \in \Sigma^*$, there is a path labelled $\sigma$ starting at $p$ and ending in a final state if and only if there is a path starting in $q$ labelled $\sigma$ which ends at a final state). We solve this problem by finding out which pair of states $p, q$ are *not* equivalent: there is a string $\sigma \in \Sigma^*$ and a path labelled $\sigma$ starting at $p$ which ends in a final state, while there is no such path for $q$ (or viceversa). We then say such $p$ and $q$ are *distinguishable*.

Notice that if $p, q$ are distinguishable, then for any pair of states $p', q'$ for which there is an input symbol $a$ such that $\delta(p', a) = p$ and $\delta(q', a) = q$, $p', q'$ are distinguishable as well (if $\sigma$ distinguishes $p$ from $q$, $a\sigma$ distinguishes $p'$ and $q'$). Clearly, the empty string distinguishes final states from non-final ones.

The algorithm uses two data structures:

- **boolean**[ ][ ]`Distinguished`: a boolean array indexed by pair of states, which records whether they are distinguished.
- (**int**[ ][ ])[ ][ ]`Pending`: array indexed by pair of states such that `Pending`$[p][q]$ is an array (or list) of pair of states $p'$,$q'$ with the property:

if `Distinguished`$[p][q]$ then `Distinguished`$[p'][q']$

Notice that `Distinguished`$[p][q]$ if and only if `Distinguished`$[q][p]$, and `Distinguished`$[p][p] =$ **false**, so we only need information for `Distinguished`$[p][q]$ for $p < q$. We express the algorithm in pseudo-algorithmic notation. Translation to actual Java code is fairly straightforward:

```
// initialization
for p ∈ F and q ∈ Q − F: Distinguished[p][q] = true;
for p, q ∈ (F × F) || (Q − F × Q − F): Distinguished[p][q] = false;
//iteration
for p, q ∈ (F × F) || (Q − F × Q − F) with p ≠ q:
    for a ∈ Σ:
        int s = Delta[p][a];
        int t = Delta[q][a];
        if Distinguished[s][t]
            then
                // mark p, q as distinguished
                Distinguished[p][q] = true;
                (* recursively mark all pairs in Pending[p][q] and
                in all the lists Pending[s][t] for the pairs s, t that get marked here *)
            else
                // no pair s, t reachable by a single input symbol
                // from p, q is distinguished
                for a ∈ Σ:
                    int s = Delta[p][a];
                    int t = Delta[q][a];
                    if s ≠ t then add [p][q] to Pending[s][t]
```

At the exit of the outermost loop, those pairs of states for which `Distinguished`$[p][q] =$ **false** can be collapsed (since we have failed to distinguish them), therefore reducing the number of states. The definition of the transition matrix `Delta`$[\bar{p}][\bar{q}]$ for the equivalence classes of $p$ and $q$ respectively, is given by that of `Distinguished`$[p][q]$. We thus get the minimized automaton.

## References

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Co., Reading, Mass., 1979.