**Minimization of Deterministic Finite State Machines**

We consider **deterministic** [finite state machine](http://lara.epfl.ch/w/finite_state_machine) $M = (\Sigma,Q,\delta,q_0,F)$.

**Goal:** build a state machine $M'$with the least number of states that accepts the language $L(M)$.

* we obtain a space-efficient, executable representation of a regular language

This is the process of *minimization* of $M$.

* an easy case of minimizing size of ‘generated code’ in compiler

We say that state machine $M$distinguishes strings $w$and $w'$iff it is not the case that ( $w \in L(M)$iff $w' \in L(M)$).

**Minimization Algorithm**

**Step 1: Remove unreachable states**

We first discard states that are not reachable from the initial state–such states are useless. In resulting machine, for each state $q$there exists a string $s$such that $\delta(q_0,s)=q$, let $s_q$one such string of minimal length.

**(Main) Step 2: Compute Non-Equivalent States**

We wish to merge states $q$and $q'$into same group as long as they “behave the same” on all future strings $w$, i.e.   
\begin{displaymath}
   \delta(q,w) \in F \mbox{ iff } \delta(q',w) \in F  \ \ \ (*)
\end{displaymath}  
for all $w$.

If the condition $(*)$above holds, we called states **equivalent**. If the condition does **not** hold, we call states $q$, $q'$**non-equivalent**.

States $q$and $q'$are $w$-non-equivalent if it is not the case that ( $\delta(q,w) \in F \mbox{ iff } \delta(q',w) \in F$).

Two states are non-equivalent iff they are $w$-non-equivalent for some string $w$.

Observe that

1. if $q \in F$and $q' \notin F$then $q$and $q'$are $\epsilon$-non-equivalent
2. if $q$and $q'$are $w$-non-equivalent and we have $\delta(r,a)=q$, $\delta(r',a)=q'$for some symbol $a \in \Sigma$, then $r$and $r'$are $aw$-non-equivalent
3. conversely, if $r$and $r'$are $w'$-non-equivalent and $w$is not an empty string, then for $w'=aw$the states $\delta(r,a)$and $\delta(r',a)$are $w$-non-equivalent

These observations lead to an iterative algorithm for computing non-equivalence relation $\nu$

1. initially put $\nu = (Q \cap F) \times (Q \setminus F)$(only final and non-final states are initially non-equivalent)
2. repeat until no more changes: if $(r,r') \notin \nu$and there is $a \in \Sigma$such that $(\delta(r,a),\delta(r',a)) \in \nu$, then

\begin{displaymath}
   \nu := \nu \cup \{(r,r')\}
\end{displaymath}

**Step 3: Merge States that are not non-equivalent**

Relation $Q^2 \setminus \nu$is an equivalence relation $\sim$. We define the ‘factor automaton’ by merging equivalent states:

* the initial state is ${q_0}/_{\sim}$
* $Q/_{\sim} = \{ \{ y \mid x \sim y \} \mid x \in Q \}$
* $F/_{\sim} = \{ \{ y \mid x \sim y \} \mid x \in F \}$
* relation $r = \{ ([x],[y]) \mid (x,y) \in \delta \}$is a function, and we can use it to define a new deterministic automaton (there is a transition in the resulting automaton iff there is a transition between two states in the original automaton)

This is the minimal automaton.

**Correctness of Constructed Automaton**

Clearly, this algorithm terminates because in worst case all states become non-equivalent. We will prove below that the resulting value $\nu$is the non-equivalence relation, i.e. the complement of relation given by $(*)$above.

By induction, we can easily prove that if $(q,q') \in \nu$, then $q$and $q'$are non-equivalent. Similarly we can show that if $q$and $q'$are $w$-non-equivalent for $w$of length $k$, then $(q,q') \in \nu$by step $k$of the algorithm. Because the algorithm terminates, this completes the proof that $\nu$is the non-equivalence relation.

Consequently, $Q^2 \setminus \nu$is the equivalence relation. From the definition of this equivalence it follows that if two states are equivalent, then so is the result of applying $\delta$to them. Therefore, we have obtained a well-defined deterministic automaton.

**Minimality of Constructed Automaton**

Note that if two distinct states are non-equivalent, there is $w$such that states $\delta(q_0,s_q w)$and $\delta(q_0,s_{q'} w)$have different acceptance, so $M$distinguishes $s_q w$and $s_{q'}w$. Now, if we take any other state machine $M' = (\Sigma,Q',\delta',q'_0,F')$with $L(M')=L(M)$, it means that $\delta'(q'_0,s_q) \neq \delta'(q'_0,s_{q'})$, otherwise $M'$would not distinguish $s_q w$and $s_{q'} w$. So, if there are $K$pairwise non-equivalent states in $M$, then a minimal finite state machine for $L(M)$must have at least $K$states. Note that if the algorithm constructs a state machine with $K$states, it means that $Q^2 \setminus \tau$had $K$equivalence relations, which means that there exist $K$non-equivalent states. Therefore, any other deterministic machine will have at least $K$states, proving that the constructed machine is minimal.

**Example**

Construct automaton recognizing

* language {=,<=}
* language {=,<=,==}

Minimize the automaton.