6.5.2

Stating and proving the Myhill-Nerode Theorem

Equivalences over automatas...

Claim (Just proved)

Let x, y be two distinct strings.
$x \equiv_L y \iff x, y$ *are indistinguishable for* L.

Corollary

If $\equiv_L$ is finite with n equivalence classes then there is a fooling set F of size n for L.

Corollary

If $\equiv_L$ has infinite number of equivalence classes $\implies \exists$ infinite fooling set for L.
$\implies L$ *is not regular.*

Equivalences over automatas...

Claim (Just proved)

Let x, y be two distinct strings.
$x \equiv_L y \iff x, y$ *are indistinguishable for* L.

Corollary

If $\equiv_L$ *is finite with* n *equivalence classes then there is a fooling set* F *of size* n *for* L.

Corollary

If $\equiv_L$ *has infinite number of equivalence classes* $\implies \exists$ *infinite fooling set for* L.
$\implies L$ *is not regular.*

Equivalences over automatas...

Claim (Just proved)

Let x, y *be two distinct strings.*
$x \equiv_L y \iff x, y$ *are indistinguishable for* L.

Corollary

If $\equiv_L$ *is finite with* n *equivalence classes then there is a fooling set* F *of size* n *for* L.

Corollary

If $\equiv_L$ *has infinite number of equivalence classes* $\implies \exists$ *infinite fooling set for* L.
$\implies L$ *is not regular.*

Equivalence classes as automata

Lemma

For all $x, y \in \Sigma^*$, *if* $[x]_L = [y]_L$, *then for any* $a \in \Sigma$, *we have* $[xa]_L = [ya]_L$.

Proof.

$[x] = [y] \implies \forall w \in \Sigma^*\colon\ xw \in L \iff yw \in L$

$\implies \forall w' \in \Sigma^*\colon\ xaw' \in L \iff yaw' \in L \qquad // \ w = aw'$

$\implies [xa]_L = [ya]_L.$ $\square$

Equivalence classes as automata

Lemma

For all $x, y \in \Sigma^*$, *if* $[x]_L = [y]_L$, *then for any* $a \in \Sigma$, *we have* $[xa]_L = [ya]_L$.

Proof.

$[x] = [y] \implies \forall w \in \Sigma^*\colon xw \in L \iff yw \in L$

$\implies \forall w' \in \Sigma^*\colon xaw' \in L \iff yaw' \in L \qquad // \; w = aw'$

$\implies [xa]_L = [ya]_L.$ □

Equivalence classes as automata

Lemma

For all $x, y \in \Sigma^*$, *if* $[x]_L = [y]_L$, *then for any* $a \in \Sigma$, *we have* $[xa]_L = [ya]_L$.

Proof.

$[x] = [y] \implies \forall w \in \Sigma^*\colon\ xw \in L \iff yw \in L$

$\implies \forall w' \in \Sigma^*\colon\ xaw' \in L \iff yaw' \in L \qquad // \ w = aw'$

$\implies [xa]_L = [ya]_L.$ $\square$

Equivalence classes as automata

Lemma

For all $x, y \in \Sigma^*$, *if* $[x]_L = [y]_L$, *then for any* $a \in \Sigma$, *we have* $[xa]_L = [ya]_L$.

Proof.

$$[x] = [y] \implies \forall w \in \Sigma^*\colon\ xw \in L \iff yw \in L$$
$$\implies \forall w' \in \Sigma^*\colon\ xaw' \in L \iff yaw' \in L \qquad // \ w = aw'$$
$$\implies [xa]_L = [ya]_L. \qquad \square$$

Set of equivalence classes

Lemma

If L has n distinct equivalence classes, then there is a DFA that accepts it using n states.

Proof.

Set of states: $Q = [L]$

Start state: $s = [\varepsilon]_L$.

Accept states: $A = \{[x]_L \mid x \in L\}$.

Transition function: $\delta([x]_L, a) = [xa]_L$.

$M = (Q, \Sigma, \delta, s, A)$: The resulting DFA.

Clearly, M is a DFA with n states, and it accepts L. $\square$

Set of equivalence classes

Lemma

If L has n distinct equivalence classes, then there is a DFA *that accepts it using n states.*

Proof.

Set of states: $Q = [L]$

Start state: $s = [\varepsilon]_L$.

Accept states: $A = \{[x]_L \mid x \in L\}$.

Transition function: $\delta([x]_L, a) = [xa]_L$.

$M = (Q, \Sigma, \delta, s, A)$: The resulting DFA.

Clearly, M is a DFA with n states, and it accepts L. $\square$

Set of equivalence classes

Lemma

If L has n distinct equivalence classes, then there is a DFA *that accepts it using n states.*

Proof.

Set of states: $Q = [L]$
Start state: $s = [\varepsilon]_L$.
Accept states: $A = \{[x]_L \mid x \in L\}$.
Transition function: $\delta([x]_L, a) = [xa]_L$.
$M = (Q, \Sigma, \delta, s, A)$: The resulting DFA.
Clearly, M is a DFA with n states, and it accepts L. □

Set of equivalence classes

Lemma

If L has n distinct equivalence classes, then there is a DFA that accepts it using n states.

Proof.

Set of states: $Q = [L]$
Start state: $s = [\varepsilon]_L$.
Accept states: $A = \{[x]_L \mid x \in L\}$.
Transition function: $\delta([x]_L, a) = [xa]_L$.
$M = (Q, \Sigma, \delta, s, A)$: The resulting DFA.
Clearly, M is a DFA with n states, and it accepts L. $\square$

Set of equivalence classes

Lemma

If L has n distinct equivalence classes, then there is a DFA that accepts it using n states.

Proof.

Set of states: $Q = [L]$
Start state: $s = [\varepsilon]_L$.
Accept states: $A = \{[x]_L \mid x \in L\}$.
Transition function: $\delta([x]_L, a) = [xa]_L$.
$M = (Q, \Sigma, \delta, s, A)$: The resulting DFA.
Clearly, M is a DFA with n states, and it accepts L. $\square$

Set of equivalence classes

Lemma

If L has n distinct equivalence classes, then there is a DFA that accepts it using n states.

Proof.

Set of states: $Q = [L]$
Start state: $s = [\varepsilon]_L$.
Accept states: $A = \{[x]_L \mid x \in L\}$.
Transition function: $\delta([x]_L, a) = [xa]_L$.
$M = (Q, \Sigma, \delta, s, A)$: The resulting DFA.
Clearly, M is a DFA with n states, and it accepts L. $\square$

Myhill-Nerode Theorem

Theorem (Myhill-Nerode)

L is regular $\iff$ $\equiv_L$ has a finite number of equivalence classes.
If $\equiv_L$ is finite with n equivalence classes then there is a DFA M accepting L with exactly n states and this is the minimum possible.

Corollary

A language L is non-regular if and only if there is an infinite fooling set F for L.

Algorithmic implication: For every DFA M one can find in polynomial time a DFA M' such that $L(M) = L(M')$ and M' has the fewest possible states among all such DFAs.

What was that all about

Summary: A regular language L has a unique (up to naming) minimal automata, and it can be computed efficiently once any DFA is given for L.

Exercise

1. Given two DFAs M_1, M_2 describe an efficient algorithm to decide if $L(M_1) = L(M_2)$.
2. Given DFA M, and two states q, q' of M, show an efficient algorithm to decide if q and q' are distinguishable. (Hint: Use the first part.)
3. Given a DFA M for a language L, describe an efficient algorithm for computing the minimal automata (as far as the number of states) that accepts L.