State Equivalence and Minimization

DFA's may have redundant states:

(i) states not reachable

(ii) states that can't reach a final state

(iii) states that are equivalent to other states.

The fundamental idea of a state is not what happened in the past; it is what can happen in the future.

Output Function of a DFA

To ease the expression of some results in the following, we are going to use a classification function in lieu of final vs. non-final states. With each state, associate a class which is...
just a member of a finite set of symbols. For example, with the DFA's studied so far, we could associate

\[
\text{class}(q) = 1 \text{ if } q \text{ is final} \\
\text{class}(q) = 0 \text{ otherwise}
\]

But we could easily generalize to \( > 2 \) classes.

Recall the \( \hat{\delta} \) from the text:

\[
\hat{\delta}(q, \varepsilon) = q \\
\hat{\delta}(q, xa) = \delta(\hat{\delta}(q,x), a)
\]

The output function is defined by

\[
\gamma(q, x) = \text{class}(\hat{\delta}(q, x))
\]

Thus for an acceptor and the above classes, \( x \) is accepted iff \( \gamma(s, x) = 1 \) where \( s \) is the start state.
Equivalent states

In a DFA, two states $q$, $q'$ are called equivalent $q \equiv q'$ iff

$$\forall x \in \Sigma^* \quad \gamma(q, x) = \gamma(q', x)$$

[Kozen uses ~ instead of \equiv.]

So two states are equivalent iff they classify each input string the same (as if these states were the start states).

Observation \equiv is an equivalence relation, as it has the properties

reflexive $q \equiv q$ because $\gamma(q, x) = \gamma(q, x)$

symmetric $q \equiv q'$ implies $q' \equiv q$

because $\gamma(q, x) = \gamma(q', x)$ implies $\gamma(q', x) = \gamma(q, x)$
Transitive \( q \equiv q' \) and \( q' \equiv q'' \) imply \( q \equiv q'' \)

because \( \delta(q, x) = \delta(q', x) \) and \( \delta(q', x) = \delta(q'', x) \)

implies \( \delta(q, x) = \delta(q'', x) \).

**Example**

\[ \begin{array}{ccc}
\bullet & \overset{a}{\rightarrow} & \circ \\
\circ & \overset{b}{\rightarrow} & \bullet \\
\bullet & \overset{a}{\rightarrow} & \circ \\
\bigcirc & \overset{b}{\rightarrow} & \bigcirc
\end{array} \]

Here \( 0 \equiv 2 \) since \( 0 \xrightarrow{b} 1 \) and \( 2 \xrightarrow{b} 1 \)

while \( 0 \xrightarrow{a} 0 \) and \( 2 \xrightarrow{a} 0 \)

and neither is final.

**Computing \( \equiv \)**

It is convenient to represent \( \equiv \) by a partition \( P \), a set of sets of states

wherein \( q \) and \( q' \) are equivalent iff they are in

the same set of states.

For example, above, we have

\[ P = \{ \{0, 2\}, \{1\} \} \]
The partition $P$ corresponding to an equivalence relation $R$ is given by

$$R(P) \iff \exists e \in S \text{ s.t. } R(e, e)$$

And every partition determines an equivalence relation.

Fact: Every equivalence relation determines a partition.

Examples: non-relations: e.g., $\{1, 2, 3\}$, $\{2\}, \{1, 3\}$

Exhaustition: $\emptyset \subseteq S_i \subseteq S$ for all $i$.

(c) Exhaustion: $S_i \neq \emptyset \implies i \in S_i$.

(i) Exhaustion: $S_i \neq \emptyset \implies S_i \subseteq S$.

To qualify as a partition, the set of sets of elements...
Example

Relation $R = \{(a,a), (a,b), (a,c), (b,a), (b,b), (b,c),
(c,a), (c,b), (c,e), (d,d), (d,e),
(e,d), (e,e), (f,f)\}$

Partition $P = \{\{a, b, c\}, \{d, e\}, \{f\}\}$

Iterative computation of $\equiv (i.e. P)$

Define $q \equiv q' \ (k \geq 0)$

$\forall x \in \mathbb{Z}^k \mid x \leq k \Rightarrow \gamma(q, x) = \gamma(q', x)$

Observation $\forall k \geq 0 \quad \equiv_k$ is an equivalence relation

Observation $q \equiv q' \iff \forall k \geq 0 \quad q_k \equiv q'_k$

Computing $\equiv_k$

Let $P_k$ be the partition corresponding to $\equiv_k$. 

.
Observation 1: For partitions the states into their assigned classes, e.g. final, non-final. The reason is that \(|x|=0\) iff \(x=\varepsilon\) and \(\gamma(q, \varepsilon)\) is the assigned class of \(q\).

Lemma 1: \(\forall a \in \Sigma \ \forall x \in \Sigma^* \ \gamma(q, ax) = \gamma(\delta(q, a), x)\)

In other words, the class of state \(q\) given \(ax\) as input is the same as the class of state \(\delta(q, a)\) given \(x\) as input. [Prove by induction on length \(|x|\).]

Lemma 2: \(\forall k \geq 0 \ \forall q \in q' \text{ iff } q \equiv q' \land \forall a \in \Sigma \ \delta(q, a) \equiv k \delta(q', a)\)

Proof: Sequences of length \(k+1\) can be broken into a first letter \(a\) and a remainder \(y\) \(|y| \leq k\). But \(\delta(q, a) \equiv k \delta(q', a)\) is the same as saying...
\( \forall y \quad |y| \leq k \Rightarrow \delta(q,ay) = \delta(q',ay) \) \tag{1}

and \( q \equiv q' \) says \( \delta(q,\varepsilon) = \delta(q',\varepsilon) \). \tag{2}

So together (1) and (2) are the same as saying

\[ \forall x \quad |x| \leq k+1 \implies \delta(q,x) = \delta(q',x) \]

Since (2) covers \( x = \varepsilon \) and (1) covers \( 1 \leq |x| \leq k+1 \).

Lemma 2 gives us a way to compute \( \equiv_k \)

for any \( k \):

For \( k = 0 \), use Observation 1.

For \( k > 0 \), use Lemma 2, having computed \( \equiv_{k-1} \).

Usually we will compute bottom-up, starting

with \( P_0 \), then \( P_1, P_2, \ldots \).
Example

\[ P_0 = \{ \{ a,b,e,f,g,h \}, \{ c \} \} \]

\[ P_1 = \{ \{ a,e,g \}, \{ b,h \}, \{ c \}, \{ f \} \} \]

\[ P_2 = \{ \{ a,e \}, \{ g \}, \{ b,h \}, \{ c \}, \{ f \} \} \]

\[ P_3 = \{ \{ a,e \}, \{ g \}, \{ b,h \}, \{ c \}, \{ f \} \} = P_{2+1} \quad \forall l \geq 0 \]

Observation 2. If we get to a k such that \( P_k = P_{k+1} \), then \( \forall l \geq 0 \) \( P_k = P_{k+1} \)

This follows from the fact that Lemma 2 can be used to compute \( P_{k+1} \) from \( P_k \), \( k \geq 0 \).
If computing $P_{k+1}$ from $P_k$ yields $P_{k+1} = P_k$, then so will computing $P_{k+2}$ from $P_{k+1}$, etc.

**Observation 3** If $P_k = P_{k+1}$, then $P_k = P$, the partition corresponding to $\Xi$.

Now we focus on showing that a $k$ will always be reached such that $P_k = P_{k+1}$.

**Definition** A partition $P$ refines a partition $P'$ on the same set, denoted $P \sqsubseteq P'$, provided

$$\forall s \in P \exists t \in P' \quad s \subseteq t$$

**Example**

$$\sqsubseteq \{\{1,2\}, \{3,4\}, \{5,6\}\} \sqsubseteq \{\{1,2,3,4\}, \{5,6\}\}$$
Every partition refines itself.

A partition $P$ properly refines $P'$, noted $P \prec P'$, iff $P \subseteq P'$ and $P \neq P'$.

$P \subseteq P'$ iff the corresponding equivalence relations $R$ and $R'$ have the property

$$\forall a, b \ aRb \text{ implies } aR'b$$

**Example**

$$\{ \{1, 2\}, \{3, 4\}, \{5, 6\} \} \preceq \{ \{1, 2, 3, 4\}, \{5, 6\} \}$$

$P \quad P'$

$1R2$ implies $1R'2$

$3R4$ implies $3R'4$

$5R6$ implies $5R'6$

but not $2R'3$ implies $2R3$, for example.

Notation $\preceq P'$ means $P' \subseteq P$.

$P \succ P'$ means $P' \not\subseteq P$. 
Observation 4  In a DFA

For any $k$, $P_{k+1} = P_k$

This is just another way of saying

$q \equiv q'$ implies $q \equiv q'_{k+1}$

Observation 5  For partitions on a finite set:

If $P \subseteq P'$ then $|P| \geq |P'|$.

If $P \subset P'$ then $|P| > |P'|$.

Observations 4 & 5 are key to understanding why a $k$ will always be reached such that $P_k = P_{k+1}$ for a DFA.

We have from observation 4:

$P_0 \equiv P_1 \equiv P_2 \equiv \ldots$
and from observation 5

\[ |P_0| \leq |P_1| \leq |P_2| \leq \ldots \]

If \( |P_k| = |P_{k+1}| \) then \( P_k = P_{k+1} \),

because \( P_{k+1} \subseteq P_k \) always, and if \( P_{k+1} \not\subseteq P_k \)

then \( |P_{k+1}| > |P_k| \), by observation 5.

**Lemma 3**

\[ \forall k \geq 0 \quad \text{If } |P_k| \leq k+1 \quad \text{then } P_k = P_{k+1} \]

**Proof**

**Basis**

For \( k=0 \), we are saying

If \( |P_0| \leq 1 \), then \( P_0 = P_1 \)

If \( |P_0| \leq 1 \) then \( |P_0| = 1 \). This can only be

the case if all states have the same assigned

class. In this case, all next states also have

the very same class. Thus the computation
of $P_1$ will yield $P_1 = P_6$.

So if $P_6 \neq P_1$ then $|P_6| > 1 = 0 + 1$.

**Induction Step**

We assume $|P_k| \leq k+1$ implies $P_k = P_{k+1}$.

We want to show $|P_{k+1}| \leq k+2$ implies $P_{k+1} = P_{k+2}$.

We'll prove the contrapositive:

$P_{k+1} \neq P_{k+2}$ implies $|P_{k+1}| > k+2$

Suppose $P_{k+1} \neq P_{k+2}$. Then $P_k \neq P_{k+1}$ (from Obs. 2).

So by the induction hypothesis, $|P_k| > k+1$.

Also, from $P_k \neq P_{k+1}$, $|P_{k+1}| > |P_k|$.

So $|P_{k+1}| > |P_k| > k+1$, giving $|P_{k+1}| > k+2$ because of the strict inequality.

**Cor 1** If $P_k \neq P_{k+1}$ then $|P_k| \geq k+2$.
Intuition: In the sequence (proper refinements)

\[ P_0 \supset P_1 \supset P_2 \supset \ldots \]

the sizes must be at least

\[ 2 < 3 < 4 < \ldots \]

Observation 6: For an n-state DFA

\[ \forall k \quad |P_k| \leq n, \]

The extreme case is \( |P_k| = n \) with each set being a singleton.

Observation 7: For an n-state DFA

if \( |P_k| = n \) then \( P_k = P_{k+1} \).

If \( |P_k| = n \), each set has one element and thus \( P_k \) cannot be further properly refined.
Theorem 1. In an $n$-state DFA

\[ \exists k \leq n-2 \quad P_k = P_{k+1} \]

Proof. It is sufficient to show $P_{n-2} = P_{n-1}$.

Suppose $P_{n-2} \neq P_{n-1}$. By Cor 1, $|P_{n-2}| \geq (n-2)+2$,

\[ |P_{n-2}| > n \quad \text{but} \quad |P_i| \leq n \quad \text{for all} \quad i \]

so $|P_{n-2}| = n$ so $P_{n-2} = P_{n-1}$

by Observation 7.
Creating The Minimal State Machine

Once $\mathcal{E}$ is known, a minimal state machine can be constructed as follows:

- The equivalence classes of $\mathcal{E}$ are the states of the new machine.
- The assigned classes of each equivalence class (e.g., final, non-final) are classes assigned to the members of the equivalence class (all must be the same in a given equivalence class).
- The start state of the new machine is the class containing the original start state.
The transition function \( \delta^* \) of the new machine is defined by:

\[
\forall q \in Q \forall a \in \Sigma \quad \delta^*([q], a) = \left[ \delta(q, a) \right] (x)
\]

We need to show that \( \delta^* \) is well defined: that \( \delta^*([q], a) \) does not depend on the particular \( q \) chosen to represent the class.

Suppose \( q \equiv q' \), we want \( \delta^*([q], a) = \delta^*([q'], a) \).

But by definition of \( \equiv \), \( \delta(q, a) \equiv \delta(q', a) \).

So the rhs of (x) is the same, whether we use \( q \) or \( q' \).

The new machine is denoted \( M/\equiv \) where \( M \) is the original machine, and is called the "quotient of \( M \) modulo \( \equiv \)."
Example

\[ P = \{ \{a, e\}, \{g\}, \{b, h\}, \{c\}, \{f\}\} \]

\[ M/\Sigma : \]

\[ \rightarrow \{a, e\} \rightarrow \{b, h\} \rightarrow \{c\} \rightarrow \{g\} \]

Note that transitions are defined consistently.
Algebraic aside

\[ M/\Xi \text{ is a homomorphic image of } M:\]

\[ h : Q \to 2^Q \quad \forall q \quad h(q) = [q] \]

Thm

\[ M/\Xi \text{ is equivalent to } M, \text{ and} \]

no two distinct states of \( M/\Xi \) are equivalent.

Proof

Show by induction

\[ \forall x \in \Sigma^* \quad \forall q \quad \delta^*(\[q\], x) = [q'] \]

iff \( \delta(q, x) \in [q'] \)

It then follows that

\[ \forall x \in \Sigma^* \quad \forall q \quad \delta^*([q], x) = \delta(q, x) \]

because \( \text{class}([q']) = \text{class}(q') \) by definition.

In particular, \( \forall x \quad \delta^*([s], x) = \delta(s, x) \)

so the two machines are equivalent.
M_MARK HILL - NERODE EQUIVALENCE

For any language $L \subseteq \Sigma^*$ we can associate the characteristic function $\varphi_L : \Sigma^* \rightarrow \{0,1\}$ defined by

$$\varphi_L(x) = \begin{cases} 1 & \text{ if } x \in L \\ 0 & \text{ otherwise} \end{cases}$$

Define binary relation $\equiv_L$ on $\Sigma^*$ for language $L$ as follows:

$$\forall x, y \in \Sigma^* \quad x \equiv_L y \iff \forall z \in \Sigma^* \quad \varphi_L(xz) = \varphi_L(yz)$$

$\equiv_L$ should look similar to the definition of $\equiv$ for a DFA.

However, $L$ is not necessarily regular.
Example 1

"Construct" $L = \{0^*10\}$:

Any two strings in $\{03^4\}$ are equivalent.

\[\{03^4\} \equiv \{13^4\}\]

But two strings in different ones of these sets are not equivalent. For example

$00 \neq_L 01$

because we can find a $z$ such that

$\varphi_L(00z) \neq \varphi_L(01z)$

namely $z = 10$. 
Example 2

"Construct" $\equiv_L$ for $L = \{0^n1^n \mid n \geq 0\}$

$0 \not\equiv 00$

$00 \not\equiv 000$

etc.

$01 \equiv 0011$

$0011 \equiv 000111$

etc.

Observation $1 \equiv_L$ is an equivalence relation

Lemma $1 \equiv_L$ has the property

$x \equiv_L y$ iff $\forall z \in \Sigma^* xz \equiv_L yz$

Proof ($\Leftarrow$) Obvious. Take $z = \varepsilon$. 
(⇒) Suppose \( x \equiv_L y \). We want to show

\[
\forall z \in \Sigma^* \quad xz \equiv_L yz
\]

Let \( z \) be an arbitrary element of \( \Sigma^* \).

To show \( xz \equiv_L yz \), we need to show

\[
(\forall w \in \Sigma^*) \quad \varphi_L(xz)w = \varphi_L(yz)w.
\]

for arbitrary \( w \).

But \( x(z)w = x(zw) \) and \( y(zw) \).

By assumption that \( x \equiv_L y \), \( \varphi_L(x(zw)) = \varphi_L(y(zw)) \).

So \( \varphi_L(xz)w = \varphi_L(yz)w \). Hence \( xz \equiv_L yz \).

**Terminology**: An equivalence relation with the above property is called a **right-congruence**.

Prove for yourself \( \equiv_L \) is a right congruence

\[
\text{iff } \forall a \in \Sigma \quad \varphi_L(xa) = \varphi_L(ya)
\]

[or we could use this as the definition and prove the other.]
Equivalence Classes of $\Xi_L$ are "Abstract States"

Analogous the construct of $M/\Xi$

for a DFA $M$, we could construct an

acceptor for any language $L$ based on $\Xi_L$.

The acceptor might not be finite-state however.

**Example 1**  
$L = \Sigma_0^* \epsilon \Sigma_0^*$

Here is the DFA with equivalence classes of

$\Xi_L$ as states:

\[
\begin{array}{c}
\circ & \rightarrow & [\epsilon] & \rightarrow & [1] & \rightarrow & [0] \\
& 1 & & & & 0,1 \\
1 & 0,1 & & & & \\
\end{array}
\]
Example 2  \( L = \{0^n1^n \mid n \geq 0\} \)

This example has an infinite set of equivalence classes, so we can only suggest the structure.

\[
\begin{array}{c}
[\varepsilon] \xrightarrow{1} [1] \xleftarrow{0,1}
\\
\downarrow \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \downarrow
\\
[0] \quad \xrightarrow{1} [01]
\\
\downarrow \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \downarrow
\\
[00] \quad \xrightarrow{1} [001]
\\
\downarrow \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \downarrow
\\
[000] \quad \xrightarrow{1} [0001]
\\
\downarrow \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \downarrow
\\
\vdots
\end{array}
\]

(Myhill–Nerode)

Theorem. A language \( L \) is regular iff \( \Xi_L \) has a finite number of equivalence classes (i.e. a finite set of abstract states).