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 q associate a class $class(q)$ which is just a member of a finite set of symbols.

For example, with the DFA's studied so far,

we could associate

$$\text{class}(q) = 1 \quad \text{if} \quad q \text{ is final}$$

$$\text{class}(q) = 0 \quad \text{otherwise}$$

But we could easily generalize to > 2 classes.

Recall the $\hat{\delta}$ from the text:

$$\hat{\delta}(q, \varepsilon) = q$$

$$\hat{\delta}(q, xa) = \delta(\hat{q}(q, x), a)$$

The <u>output function</u> 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

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

[Kozen uses $\approx$ instead of $\equiv$.]

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

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

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

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

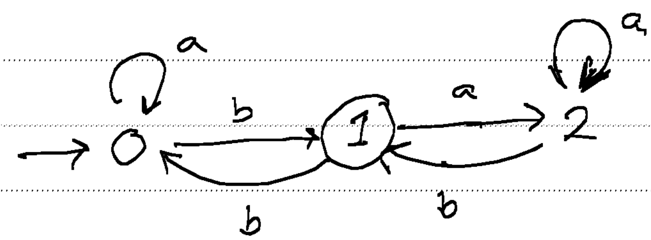
because $\delta(q, x) = \delta(q', x)$ implies $\delta(q', x) = \delta(q, x)$

transitive $q \equiv q'$ and $q' \equiv q''$ imply $q \equiv q''$

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

imply $\delta(q, x) = \delta(q'', x)$.

Example



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

$$P = \{\{0, 2\}, \{1\}\}$$

To qualify as a <u>partition</u>, the set of sets of states

$\{s_1, s_2, \ldots, s_m\}$ must have these properties:

(1) Exclusion $\quad s_i \cap s_j \neq \emptyset$ implies $s_i = s_j$

(2) Exhaustion $\quad \bigcup_i s_i =$ all states

<u>Examples</u> non-partitions: $\{\{0,1\},\{1,2\}\}$, $\{\{\},\{1,2\}\}$ partition: $\{\{1,2\}\}$

<u>Fact</u> Every equivalence determines a partition and every partition determines an equivalence relation.

The equivalence relation R corresponding to partition P is given by

$$a R b \quad \text{iff} \quad \exists s \in P\ (a \in s \wedge b \in s)$$

You should check that R has the desired properties.

The partion P corresponding to an equivalence relation R is $P = \{\{b \mid aRb\} \mid b \in \text{domain}(R)\}$.

Example

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

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

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

Define $q \underset{k}{\equiv} q'$ $(k \geq 0)$

$\forall x \in \Sigma^* \quad |x| \leq k \Rightarrow \delta(q,x) = \delta(q',x)$

Observation $\forall k \geq 0$ $\underset{k}{\equiv}$ is an equivalence relation

Observation $q \equiv q'$ iff $\forall k \geq 0$ $q \underset{k}{\equiv} q'$

Computing $\underset{k}{\equiv}$ Let P_k be the partition corresponding to $\equiv_k$.

<u>Observation 1</u> P_0 partitions the states into their <u>assigned</u> classes, e.g. final, non-final. The reason is that $|x|=0$ iff $x=\varepsilon$ and $\gamma(q,\varepsilon)$ <u>is</u> the assigned class of q.

<u>Lemma 1</u> $\forall a \in \Sigma \;\; \forall x \in \Sigma^* \;\; \gamma(q, ax) = \gamma(\delta(q,a), x)$

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).]

<u>Lemma 2</u> $\forall k \geq 0 \quad q \underset{k+1}{\equiv} q'$ iff

$$q \underset{0}{\equiv} q' \wedge \forall a \in \Sigma \quad \delta(q,a) \underset{k}{\equiv} \delta(q',a)$$

<u>Proof</u> Sequences of length $k+1$ can be broken into a first letter a and a remainder y, $|y| \leq k$. But $\delta(q,a) \underset{k}{\equiv} \delta(q',a)$ is the same as saying

$$\forall y \quad |y| \le k \rightarrow \gamma(q, ay) = \gamma(q', ay) \qquad (1)$$

and $q \underset{0}{\equiv} q'$ says $\gamma(q, \varepsilon) = \gamma(q', \varepsilon)$. (2)

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

$$\forall x \quad |x| \le k+1 \longrightarrow \gamma(q, x) = \gamma(q', x)$$

since (2) covers $x = \varepsilon$ and (1) covers $1 \le |x| \le k+1$.

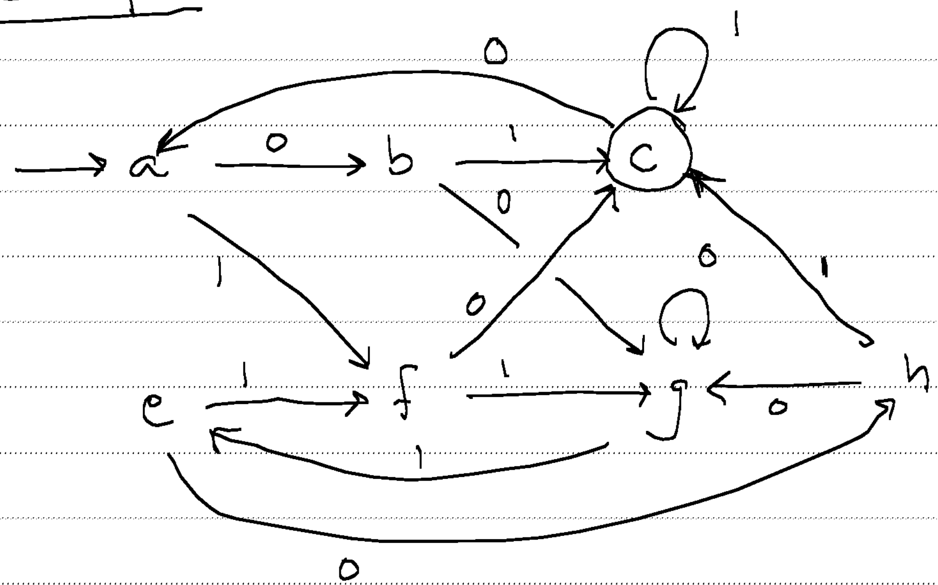
Lemma 2 gives us a way to compute $\underset{k}{\equiv}$ for any k:

For $k=0$, use Observation 1.

For $k>0$, use lemma 2, having computed $\underset{k-1}{\equiv}$.

Usually we will compute bottom-up, starting with P_0, then P_1, P_2, ...

Example



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

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

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

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

Observation 2 If we get to a k such that

$P_k = P_{k+1}$, then $\forall l \geq 0 \quad P_k = P_{k+l}$

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 $\equiv$.

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

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

Every partition refines itself.

A partition P <u>properly refines</u> P', noted $P \sqsubset P'$,

iff $P \sqsubseteq P'$ and $P \neq P'$.

$P \sqsubseteq P'$ iff the corresponding equivalence

relations R and R' have the property

$\forall a, b \quad aRb$ implies $aR'b$

<u>Example</u>

$$\underbrace{\{ \{1,2\}, \{3,4\}, \{5,6\} \}}_{P} \quad \sqsubseteq \quad \underbrace{\{ \{1,2,3,4\}, \{5,6\} \}}_{P'}$$

$1 R 2$ implies $1 R' 2$

$3 R 4$ implies $3 R' 4$

$5 R 6$ implies $5 R' 6$

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

<u>Notation</u> $P \sqsupseteq P'$ means $P' \sqsubseteq P$.

$P \sqsupset P'$ means $P' \sqsubset P$.

Observation 4 In a DFA

$$\forall k \quad P_{k+1} \sqsubseteq P_k$$

This is just another way of saying

$$q \underset{k+1}{\equiv} q' \text{ implies } q \underset{k}{\equiv} q'$$

Observation 5 For partitions on a finite set:

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

If $P \sqsubset P'$ 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:

$$P_0 \sqsupseteq P_1 \sqsupseteq P_2 \sqsupseteq \ldots$$

and from observation 5

$$|P_0| \le |P_1| \le |P_2| \le \ldots$$

If $|P_k| = |P_{k+1}|$ then $P_k = P_{k+1}$,

because $P_{k+1} \sqsubseteq P_k$ always, and if $P_{k+1} \underset{\neq}{\sqsubset} P_k$

then $|P_{k+1}| > |P_k|$, by observation 5.

Lemma 3

$$\forall k \ge 0 \quad \text{If } |P_k| \le k+1 \text{ then } P_k = P_{k+1}.$$

Proof

Basis

For $k=0$, we are saying

$$\text{If } |P_0| \le 1, \text{ then } P_0 = P_1$$

If $|P_0| \le 1$ 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 $P_1 = P_0$.

So if $P_0 \neq P_1$ then $|P_0| > 1 = 0+1$.

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:

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

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 $|P_k| \geq k+2$

<u>Intuition</u> In the sequence (<u>proper</u> refinements)

$$P_0 \sqsupset P_1 \sqsupset P_2 \sqsupset \dots$$

the sizes must be <u>at least</u>

$$2 < 3 < 4 < \dots$$

<u>Observation 6</u> For an n-state DFA

$\forall k \quad |P_k| \le n.$

The extreme case is $|P_k| = n$ with each set being a singleton.

<u>Observation 7</u> 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

$$\exists k \le n-2 \qquad P_k = P_{k+1}$$

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

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

$|P_{n-2}| \geq n$, but $|P_i| \leq n$ for all i

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

by Observation 7.

Creating The Minimal State Machine

Once $\equiv$ is known, a minimal state machine can be constructed as follows:

- The equivalence classes of $\equiv$ 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 δ^* of the new machine is defined by:

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

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

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

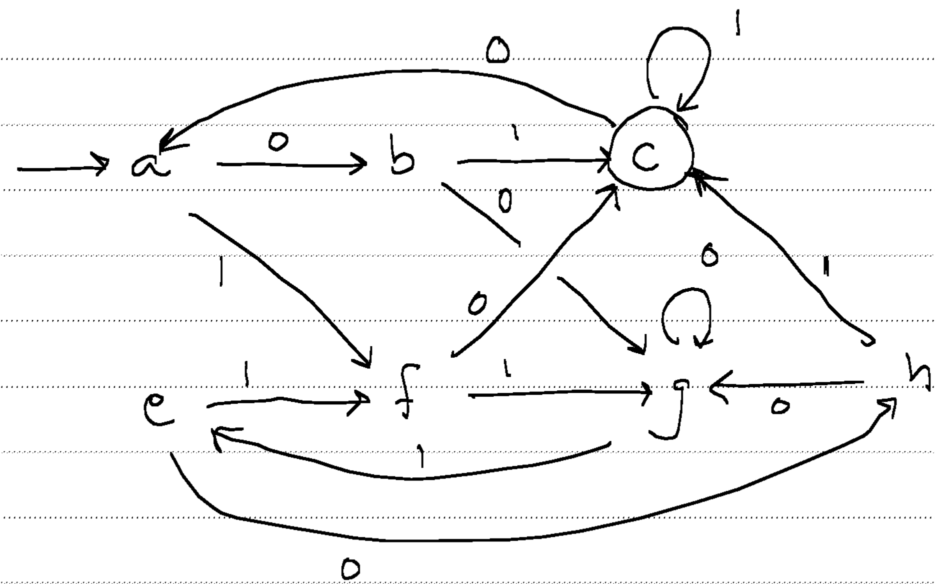
But by definition of $\equiv$, $\delta(q, a) \equiv \delta(q', a)$.

So the rhs of (*) 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 <u>"quotient of M modulo $\equiv$"</u>.

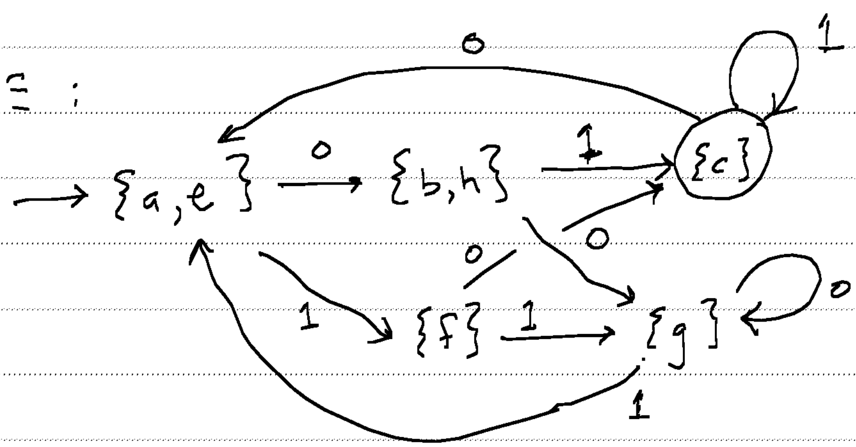
Example

M



$$P^* = \{\, \{a\,e\},\ \{g\},\ \{b\,h\},\ \{c\},\ \{f\} \,\}$$

$M/\equiv$:



Note that transitions are defined consistently.

Algebraic aside

$M/\equiv$ is a homomorphic image of M:

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

Thm

$M/\equiv$ is equivalent to M, and

no two distinct states of $M/\equiv$ are equivalent.

Proof

Show by induction

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

$$\text{iff} \quad \delta(q, x) \in [q']$$

It then follows that

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

because class$([q'])$ = class(q') by definition.

In particular, $\forall x \ \gamma^*([s], x) = \gamma(s, x)$

so the two machines are equivalent.

Myhill-Nerode Equivalence

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

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

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

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

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

However, L is not necessarily regular.

Example 1

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

Any two strings in $\{0\}^*$ are equivalent.

" $\{0\}^*\{1\}$ "

$\{0\}^*\{10\}$ "

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

$$00 \not\equiv_L 01$$

because we can find a z such that

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

namely $z = 10$.

Example 2

"Construct" $\equiv_L$ for $L = \{0^n 1^n \mid n \geq 0\}$

$0 \not\equiv 00$

$00 \not\equiv 000$

etc.

$01 \equiv 0011$

$0011 \equiv 000111$

etc.

Observation 1 $\equiv_L$ is an equivalence relation

Lemma 1 $\equiv_L$ has the property

$$x \equiv_L y \quad \text{iff} \quad \forall z \in \Sigma^* \quad xz \equiv_L yz$$

Proof ($\Leftarrow$) Obvious. Take $z = \varepsilon$.

($\Rightarrow$) 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 Σ^*.

To show $xz \equiv_L yz$, we need to show

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

But for arbitrary w, $(xz)w = x(zw)$ and $y(zw)$.

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

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

<u>Terminology</u> An equivalence relation with the above property is called a <u>right-congruence</u>.

<u>Prove for yourself</u> $\equiv_L$ is a right congruence iff $\forall a \in \Sigma$, $\varphi_L(xa) = \varphi_L(ya)$

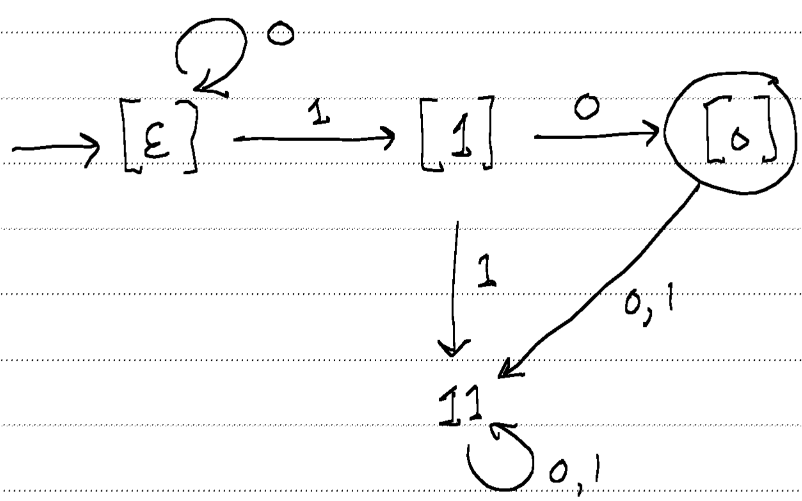
[or we could use this as the definition, and prove the other.]

Equivalence Classes of $\equiv_L$ are "Abstract States"

Analogous the construct of $M/\equiv$ for a DFA M, we could construct an acceptor for <u>any</u> language L based on $\equiv_L$. The acceptor might not be finite-state however.

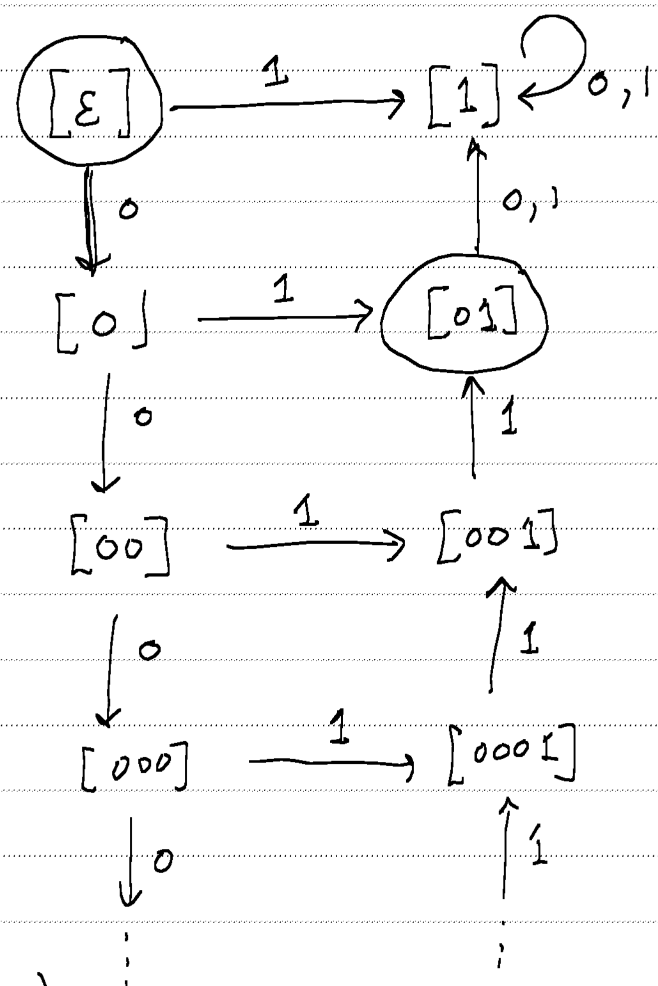
<u>Example 1</u> $L = \{0\}^* \{10\}$

Here is the DFA with equivalence classes of $\equiv_L$ as states:



Example 2 $L = \{0^n 1^n \mid n \geq 0\}$

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



(Myhill-Nerode)

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