07-0: **Fun with Finite Automata**

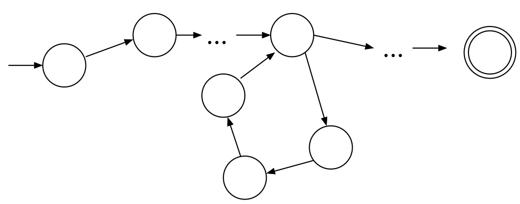
- Create a Finite Automata (DFA or NFA) for the language:
  - $L = \{0^n 1^n : n > 0\}$
  - $\{01, 0011, 000111, 00001111, \dots\}$

07-1: **Fun with Finite Automata**

- $L = \{0^n 1^n : n > 0\}$ is not regular!
- Why?
  - Need to keep track of how many 0's there are, and match 1's
  - Only way to store information in DFA is through what state the machine is in
  - Finite number of states (DFA)
  - Unbounded number of 0's before the 1's

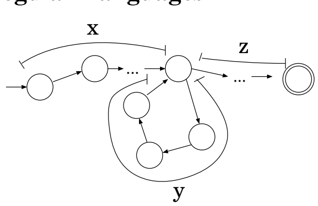
07-2: **Non-Regular Languages**

- If a DFA M has k states, and a string w accepted by M has n characters, $n > k$, computation must include a loop



- Pigeonhole Principle:
  - More transitions than states
  - Some transition must enter the same state twice

07-3: **Non-Regular Languages**



- Break string into $w = xyz$
- If $w = xyz$ is accepted, then $w' = xyyz$ will also be accepted
- If $w = xyz$ is accepted, then $w' = xyyyz$ will also be accepted
- If $w = xyz$ is accepted, then $w' = xz$ will also be accepted

07-4: **Pumping Lemma**

- If a language L is regular, then:

- $\exists n \geq 1$ such that any string $w \in L$ with $|w| \geq n$ can be rewritten as $w = xyz$ such that
  - $y \neq \epsilon$
  - $|xy| < n$
  - $xy^iz \in L$ for all $i \geq 0$

07-5: **Using the Pumping Lemma**

- Assume L is regular
- Let n be the constant of the pumping lemma
- Create a string w such that $|w| > n$
- Show that for *every* legal decomposition of $w = xyz$ such that:
  - $|xy| < n$
  - $y \neq \epsilon$

  There is an i such that $xy^iz \notin L$
- Conclude that L must not be regular

07-6: **Using the Pumping Lemma**

- Assume L is regular
- Let n be the constant of the pumping lemma
- Create a string w such that $|w| > n$
- Show that for *every* legal decomposition of $w = xyz$ such that:
  - $|xy| < n$
  - $y \neq \epsilon$

  There is an i such that $xy^iz \notin L$
- Conclude that L must not be regular

$L = \{0^n1^n : n > 0\}$

07-7: **Using the Pumping Lemma**

$L = \{0^n1^n : n > 0\}$

- Let n be the constant of the pumping lemma
- Consider the string $w = 0^n1^n$
- If we break $w = xyz$ such that $|xy| < n, |y| > 0$,
  then x and y must be all 0's
  - $x = 0^j$, $y = 0^k$, $z = 0^{n-k-j}1^n$
  - Consider $w' = xy^2z = 0^{n+k}1^n$ for some $0 < k < n$
    - $w' \notin L$
- L is not regular (by the pumping lemma)

07-8: **Using the Pumping Lemma**

- Assume L is regular
- Let n be the constant of the pumping lemma
- Create a string w such that $|w| > n$
- Show that for *every* legal decomposition of $w = xyz$ such that:
  - $|xy| < n$
  - $y \neq \epsilon$

  There is an i such that $xy^iz \notin L$
- Conclude that L must not be regular

$L = \{ww : w \in (a+b)^*\}$ 07-9: **Using the Pumping Lemma**
$L = \{ww : w \in (a+b)^*\}$

- Let n be the constant of the pumping lemma
- Consider $w = a^nba^nb \in L$
- If we break $w = xyz$ such that $|xy| < n$, $|y| > 0$,
  then x and y must be all a's
  - $x = a^j$, $y = a^k$, $z = a^{n-k-j}ba^n$
- Consider $w' = xy^2z = a^{n+k}ba^nb$. As long as $k > 0$, the first half of w' contains all a's, while the second half contains two b's. Thus w' is not of the form ww, and is not in L. Hence, L is not regular by the pumping lemma.

07-10: **Using the Pumping Lemma** You have an adversary who thinks L is regular. You need to prove that your adversary is wrong.

you	Language L is not regular!
adv	Yes it is! I have a DFA to prove it!
you	Oh really? How many states are in your DFA?
adv	n
you	OK, here's a string $w \in L$ with $|w| > n$. Your machine must accept w – but since $|w| > n$, there must be a loop in your computation. Where's the loop?
adv	Right here! (breaks w into xyz, where y is the part of the string that goes through the loop)
you	Ah hah! If we go through the loop 2 times instead of 1, we get a string not in L that your machine will accept!
adv	Drat!

07-11: **Using the Pumping Lemma**
You have an adversary who thinks L is regular. You need to prove that your adversary is wrong.

- Your adversary picks an n
- You pick a $w \in L$ (such that $|w| > n$)
- Your adversary breaks w into xyz (subject to $|xy| < n$, $|y| > 0$)
- You pick an i such that $xy^iz \notin L$

07-12: **Using the Pumping Lemma**
You have an adversary who thinks L is regular. You need to prove that your adversary is wrong.

- Your adversary picks an n
- You pick a $w \in L$ (such that $|w| > n$)
- Your adversary breaks w into xyz (subject to $|xy| < n$, $|y| > 0$)
- You pick an i such that $xy^iz \notin L$

You don't *really* have an adversary, so you need to show that for *any* n, you can create a string w, and for *any* way that w can be broken into xyz, there is an i such that $xy^iz \notin L$

07-13: **Using the Pumping Lemma**

- Assume L is regular
- Let n be the constant of the pumping lemma
- Create a string w such that $|w| > n$
- Show that for *every* legal decomposition of $w = xyz$ such that:
  - $|xy| < n$
  - $y \neq \epsilon$

  There is an i such that $xy^iz \notin L$
- Conclude that L must not be regular

$L = \{w : w \in (a^*b^*) \wedge w$ contains more a's than b's $\}$

07-14: **Using the Pumping Lemma**

$L = \{w : w \in (a^*b^*) \wedge w$ contains more a's than b's $\}$

- Let n be the constant of the pumping lemma
- Consider $w = a^nb^{n-1} \in L$
- If we break $w = xyz$ such that $|xy| < n$, $|y| > 0$,
  then x and y must be all a's
  - $x = a^j$, $y = a^k$, $z = a^{n-k-j}b^{n-1}$
- Consider $w' = xy^0z = a^{n-k}b^{n-1}$. As long as $k > 0$, w' has at least as many b's as a's, and is not in L. Hence, L is not regular, by the pumping lemma.

07-15: **Using the Pumping Lemma**

- Assume L is regular
- Let n be the constant of the pumping lemma
- Create a string w such that $|w| > n$
- Show that for *every* legal decomposition of $w = xyz$ such that:
  - $|xy| < n$
  - $y \neq \epsilon$

  There is an i such that $xy^iz \notin L$

- Conclude that L must not be regular

$L = \{w : w \in (a+b)^* \wedge w$ has an even number of a's and an odd number of b's $\}$

07-16: **Using the Pumping Lemma**

$L = \{w : w \in (a+b)^* \wedge w$ has an even number of a's and an odd number of b's $\}$

- Let n be the constant of the pumping lemma
- Consider $w = a^{2n}b \in L$
- If we break $w = xyz$ such that $|xy| < n$, $|y| > 0$,
  then x and y must be all a's
  - $x = a^j$, $y = a^k$, $z = a^{2n-k-j}b$
- As long as k is even, $w' = xy^iz \in L$ for all i

Remember, we don't get to choose how the string is broken into xyz – need to show that for *any* way the string can be broken into xyz, there exists an i such that $xy^iz \notin L$

07-17: **Using the Pumping Lemma**

$L = \{w : w \in (a+b)^* \wedge w$ has an even number of a's and an odd number of b's $\}$

- We failed to prove L is not regular. Does that mean that L must be regular?

07-18: **Using the Pumping Lemma**

$L = \{w : w \in (a+b)^* \wedge w$ has an even number of a's and an odd number of b's $\}$

- We failed to prove L is not regular. Does that mean that L must be regular?
  - No! We may not have chosen a clever enough w
  - Similarly, failing to create an NFA for a language does not prove that it is not regular.
- How can we prove that L is regular?

07-19: **Using the Pumping Lemma**

$L = \{w : w \in (a+b)^* \wedge w$ has an even number of a's and an odd number of b's $\}$

- We failed to prove L is not regular. Does that mean that L must be regular?
  - No! We may not have chosen a clever enough w
  - Similarly, failing to create an NFA for a language does not prove that it is not regular.
- How can we prove that L is regular?
  - Create a regular expression, DFA, or NFA that describes L

07-20: **Closure Properties**

Since some languages are regular, and some are not, we can consider closure properties of regular languages

- Is L_{REG} closed under union?
- Is L_{REG} closed under complementation?
- Is L_{REG} closed under intersection?

07-21: **Closure Properties**

- Is L_{REG} closed under union?

07-22: **Closure Properties**

- Is L_{REG} closed under union?

$$L_1 = L[r_1], L_2 = L[r_2]$$
$$L_1 \cup L_2 = L[(r_1 + r_2)]$$

07-23: **Closure Properties**

- Is L_{REG} closed under complementation?

Given any *DFA* $M = (K, \Sigma, \delta, s, F)$, create $M' = (K', \Sigma', \delta', s', F')$ such that $L[M'] = \overline{L[M]}$

07-24: **Closure Properties**

- Is L_{REG} closed under complementation?

Given any *DFA* $M = (K, \Sigma, \delta, s, F)$, create $M' = (K', \Sigma', \delta', s', F')$ such that $L[M'] = \overline{L[M]}$

- $K' = K$
- $\Sigma' = \Sigma$
- $\delta' = \delta$
- $s' = s$
- $F' = K - F$

07-25: **Closure Properties**

- Is L_{REG} closed under intersection?

07-26: **Closure Properties**

- Is L_{REG} closed under intersection?
  - $\overline{\overline{A} \cup \overline{B}} = A \cap B$
  - (diagram on board)
- We can also use a direct construction
  - $L_1 =$ all strings over $\{a, b\}$ that begin with aa
  - $L_2 =$ all strings over $\{a, b\}$ that end with aa
  - Construct $L_1 \cap L_2$

07-27: **Closure Properties**

Given DFA $M_1 = (K_1, \Sigma_1, \delta_1, s_1, F_1)$ and DFA $M_2 = (K_2, \Sigma_2, \delta_2, s_2, F_2)$, create DFA M such that $L[M] = L[M_1] \cap L[M_2]$

07-28: **Closure Properties**

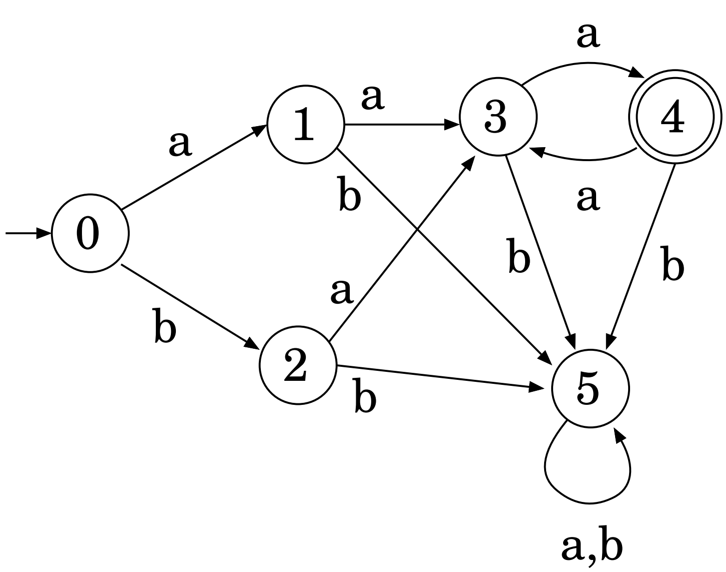
Given $M_1 = (K_1, \Sigma_1, \delta_1, s_1, F_1)$ and $M_2 = (K_2, \Sigma_2, \delta_2, s_2, F_2)$, create M such that $L[M] = L[M_1] \cap L[M_2]$

- $K = K_1 \times K_2$
- $\Sigma = \Sigma_1 = \Sigma_2$
- $\delta = \{(((q_1, q_2), a), (q_1', q_2')) : ((q_1, a), q_1') \in \delta_1, ((q_2, a), q_2') \in \delta_2\}$
- $s = (s_1, s_2)$
- $F = \{(f_1, f_2) : f_1 \in F_1, f_2 \in F_2\}$

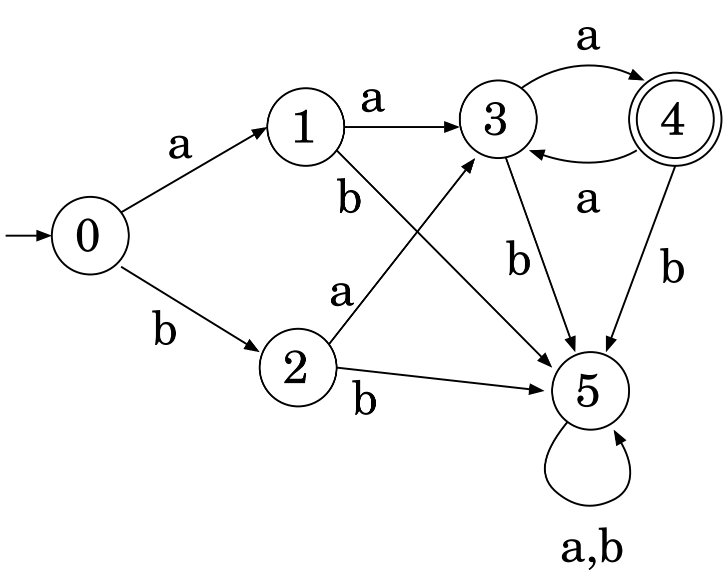
07-29: **State Minimization**

- Possible to have several different DFA that all accept the same language
- Redundant states – duplicate the effort of other states

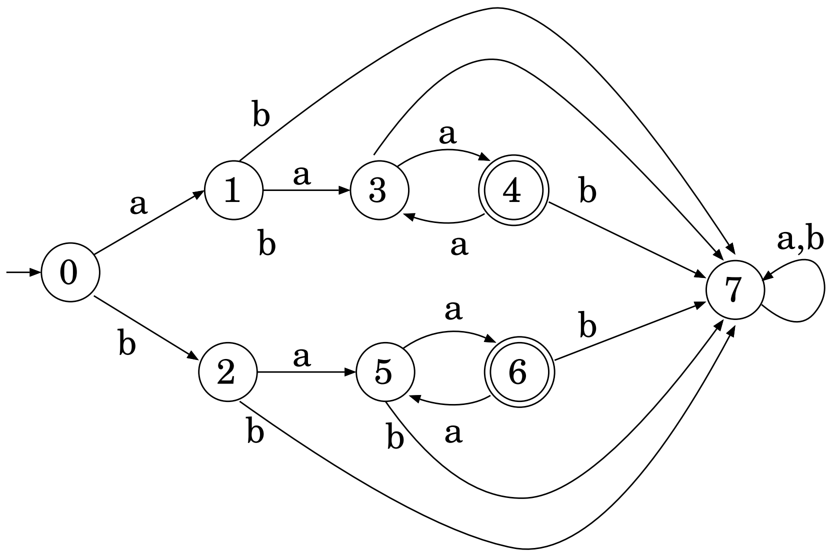
07-30: **State Minimization**



What is $L[M]$? 07-31: **State Minimization**



07-32: **State Minimization**



07-33: **State Minimization**

- Two states q_1 and q_2 are equivalent if:
  - Every string that drives q_1 to an accept state also drives q_2 to an accept state
  - Every string that drives q_2 to an accept state also drives q_1 to an accept state

07-34: **State Minimization**

- Two states q_1 and q_2 of DFA M are equivalent if:
  - $\forall w \in \Sigma^*, ((q_1, w) \mapsto_M^* (f_1, \epsilon) \wedge$
    $(q_2, w) \mapsto_M^* (f_2, \epsilon) \wedge f_1 \in F_M) \Rightarrow f_2 \in F_M$

07-35: **State Minimization**

- Two states q_1 and q_2 are equivalent with respect to a string w if and only if
  $((q_1, w) \mapsto_M^* (f_1, \epsilon) \wedge$
  $(q_2, w) \mapsto_M^* (f_2, \epsilon) \wedge f_1 \in F_M) \Rightarrow f_2 \in F_M$
  and
  $((q_1, w) \mapsto_M^* (q_3, \epsilon) \wedge$
  $(q_2, w) \mapsto_M^* (q_4, \epsilon) \wedge q_3 \notin F_M) \Rightarrow q_4 \notin F_M$
- Two states q_1 and q_2 are equivalent if they are equivalent with respect to all strings $w \in \Sigma^*$

07-36: **State Minimization**

- How do we determine if two states q_1 and q_2 are equivalent?
  - Check to see if they are equivalent with respect to strings of length 0

07-37: **State Minimization**

- How do we determine if two states q_1 and q_2 are equivalent?
  - Check to see if they are equivalent with respect to strings of length 0
  - Check to see if they are equivalent with respect to strings of length 1

07-38: **State Minimization**

- How do we determine if two states q_1 and q_2 are equivalent?
  - Check to see if they are equivalent with respect to strings of length 0
  - Check to see if they are equivalent with respect to strings of length 1
  - Check to see if they are equivalent with respect to strings of length 2
    .. and so on

07-39: **State Minimization**

- When are q_1 and q_2 equivalent with respect to all strings of length 0?

07-40: **State Minimization**

- When are q_1 and q_2 equivalent with respect to all strings of length 0?
- Both q_1 and q_2 are accept states, or neither q_1 nor q_2 are accept states

07-41: **State Minimization**

- Two states q_1 and q_2 are equivalent with respect to all strings of length n if ..
  - Hint: Think inductively

07-42: **State Minimization**

- Two states q_1 and q_2 are equivalent with respect to all strings of length n if ..
  - Hint: Think inductively
  - Hint 2: If we knew which states were equivalent with respect to all strings of length $n-1$ ...

07-43: **State Minimization**

- Two states q_1 and q_2 are equivalent with respect to all strings of length n if, for all $a \in \Sigma$
  - $((q_1, a), q_3) \in \delta$ $[\delta(q_1, a) = q_3]$
  - $((q_2, a), q_4) \in \delta$ $[\delta(q_2, a) = q_4]$
  - q_3 and q_4 are equivalent with respect to all strings of length $n-1$

07-44: **State Minimization**

- Equivalence matrix $E^{(i)}$:
  - $E^{(i)}[i, j] = 1$ iff q_i and q_j are equivalent with respect to all strings of length $\leq i$
  - Only need to calculate upper triangle of matrix (why?)
- $E^{(*)}[i, j] = 1$ iff q_1 and q_j are equivalent with respect to all strings (that is, if q_1 and q_j are equivalent)

07-45: **State Minimization**

- $E^{(0)}$:
  - $E^{(0)}[i,j] = ...$

07-46: **State Minimization**

- $E^{(0)}$:
  - $E^{(0)}[i,j] = 1$ if q_i and q_j are both accept states, or both non-accept states
  - $E^{(0)}[i,j] = 0$ if q_i is an accept state, and q_j is not an accept state
  - $E^{(0)}[i,j] = 0$ if q_i is not an accept state, and q_j is an accept state

07-47: **State Minimization**

- $E^{(n)}[i,j] = 1$ if, for all $a \in \Sigma$
  - $((q_i, a), q_k) \in \delta$ $[\delta(q_i, a) = q_k]$
  - $((q_j, a), q_l) \in \delta$ $[\delta(q_j, a) = q_l]$
  - $E^{(n-1)}[q_k, q_l] = 1$

07-48: **State Minimization**

- Creating $E^{(*)}$:
  - First, create $E^{(0)}$

for $i = 0$ to n
    for $j = (i+1)$ to n
        if $(q_i \in F \wedge q_j \in F) \vee (q_i \notin F \wedge q_j \notin F)$
            $E[i,j] = 1$
        else
            $E[i,j] = 0$

07-49: **State Minimization**
Repeat:

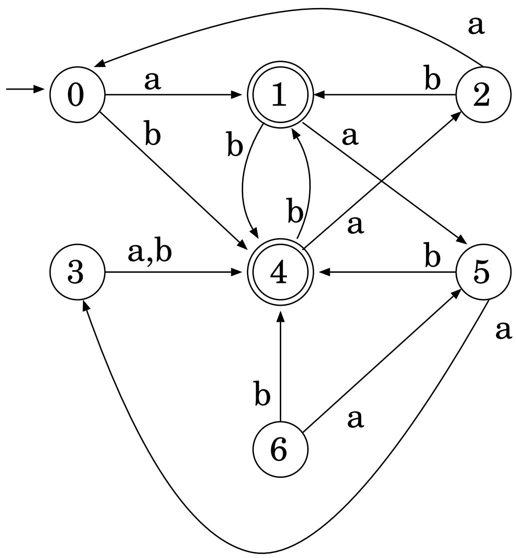
for $i = 0$ to n
    for $j = (i+1)$ to n
        for each $a \in \Sigma$
            $k = \delta(i, a)$
            $l = \delta(j, a)$
            if $E[k,l] == 0$
                set $E[i,j] = 0$

Until no changes are made 07-50: **State Minimization**

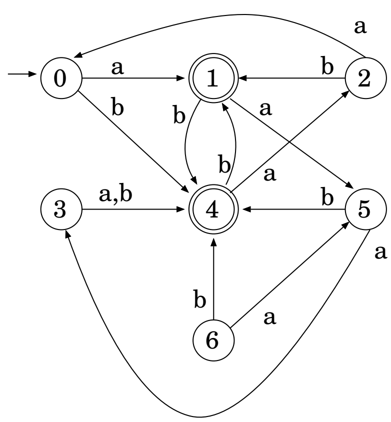
- Given any DFA M, we can create an equivalent DFA with the minimum number of states as follows:
  - Calculate $E^{(*)}$, to find equivalent states

- While there is a pair q_i, q_j of equivalent states in M
  - Change all transitions into q_j to transitions to q_i
  - Remove q_j and all transitions out of q_j
- Finally do a DFS form the initial state, and remove all states not reachable from the initial state

07-51: **State Minimization Example**



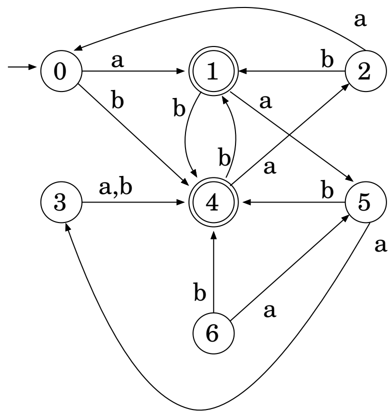
07-52: **State Minimization Example**



	0	1	2	3	4	5	6
0		0	1	1	0	1	1
1			0	0	1	0	0
2				1	0	1	1
3					0	1	1
4						0	0
5							1

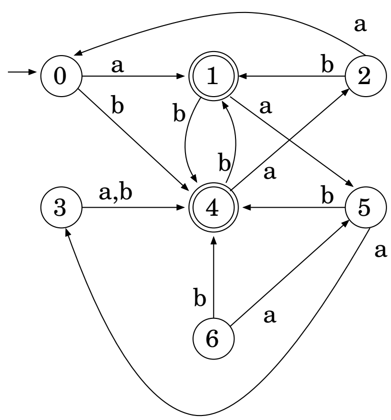
07-53: **State Minimization Example**

07-54: **State Minimization Example**

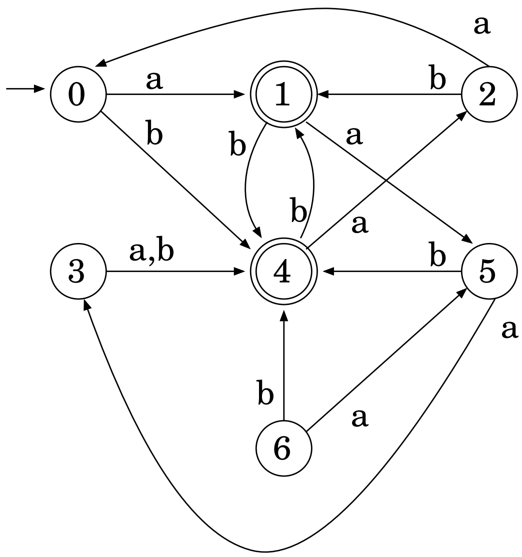


	0	1	2	3	4	5	6
0		0	0	1	0	0	0
1			0	0	1	0	0
2				0	0	1	1
3					0	0	0
4						0	0
5							1

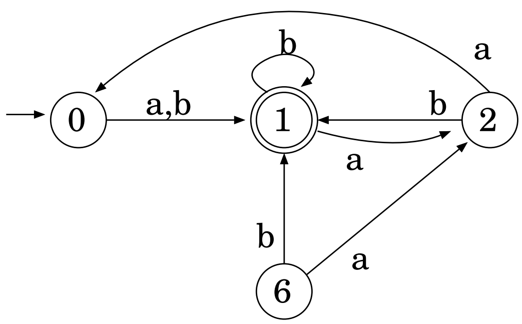
07-55: **State Minimization Example**



	0	1	2	3	4	5	6
0		0	0	1	0	0	0
1			0	0	1	0	0
2				0	0	1	0
3					0	0	0
4						0	0
5							0



07-56: **State Minimization Example**



07-57: **State Minimization Example**