

BBM402-Lecture 2: DFA's and Closure Properties

Lecturer: Lale Özkahya

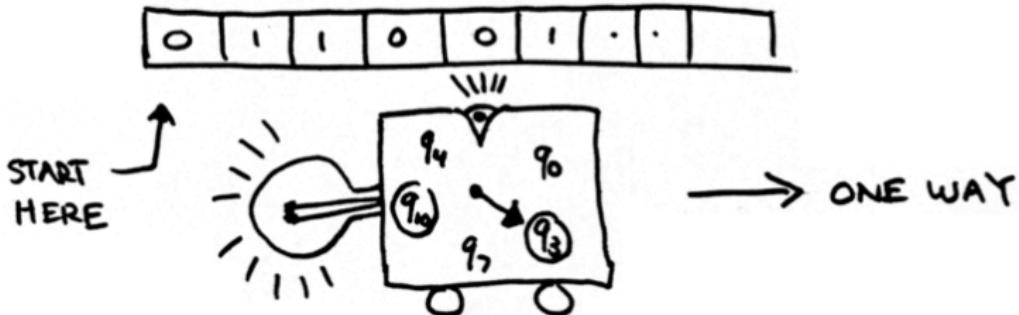
Resources for the presentation:

<http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/Syllabus/>
<https://courses.engr.illinois.edu/cs498374/lectures.html>

DFAs (*also called FSMS*)

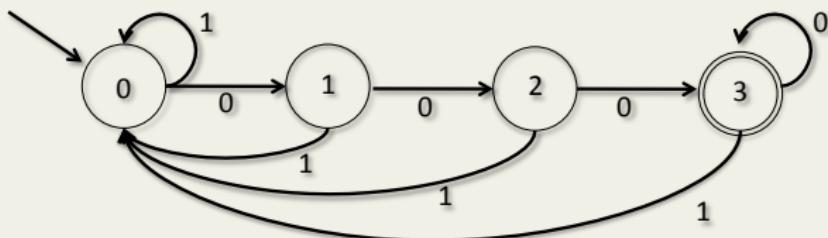
- A simple(st?) model of what a computer is
- Many devices modeled, programmed as DFAs
 - Vending machines
 - Elevators
 - Digital watch logic
 - Calculators
 - Lexical analysis part of program compilation
- Very limited, but observable universe is finite...

Typical DFA



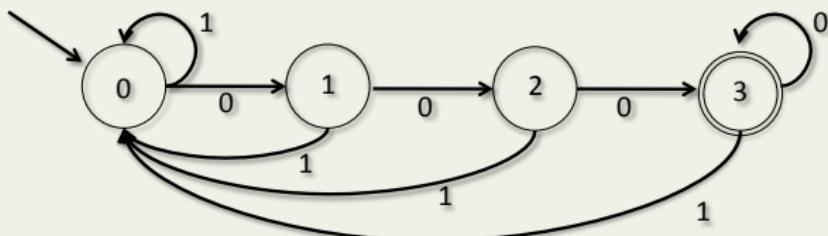
- Start state q_0
- Start at left, scan symbol, change state, move right.
- Rules of form “if in state q scanning symbol s then go to state p and move right.”
- Some states (circled) are *accepting*.
- M accepts the input string if a circled state is reached after scanning the last symbol.

Graphical Representation



- Directed graph with edges labeled with chars in Σ
- For each state (vertex) q and symbol a in Σ there is *exactly* one edge leaving q labeled with a . $q \xrightarrow{a} p$
- Accepting state(s) are double-circled
- Initial state has pointer, or is obviously labeled (0 , q_0 , “start”...)

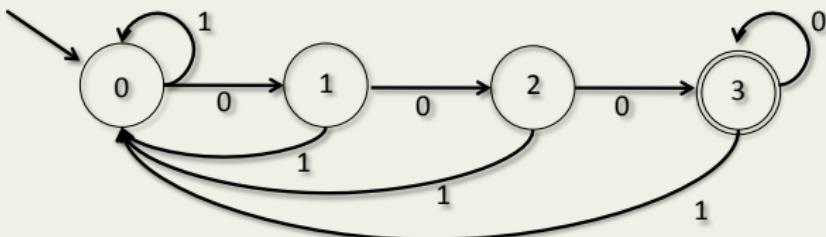
Graphical Representation



- Where does 001 lead? 10010?
- Which strings end up in accepting state?
- Prove it
- *Every string has one path* that it follows

$q \xrightarrow{a} p$ versus $q \xrightarrow{w} p$

Graphical Representation



Definition

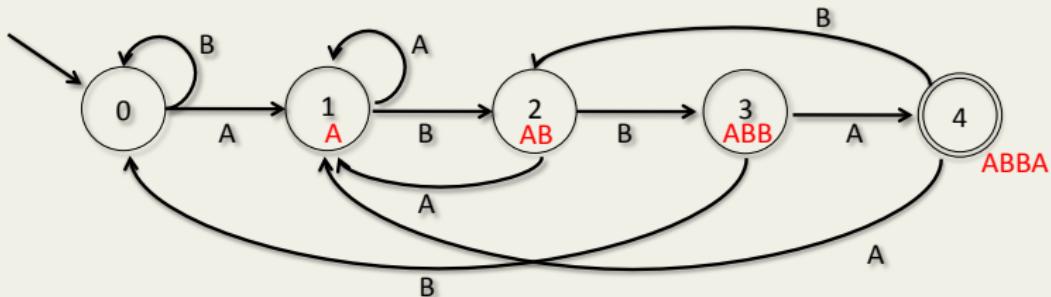
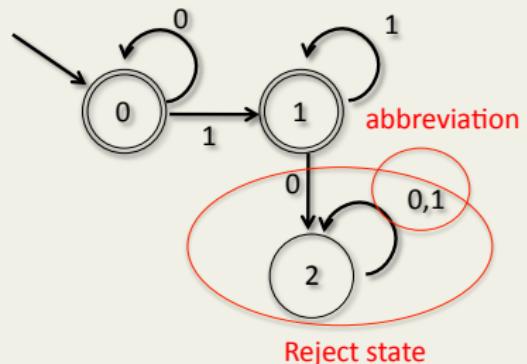
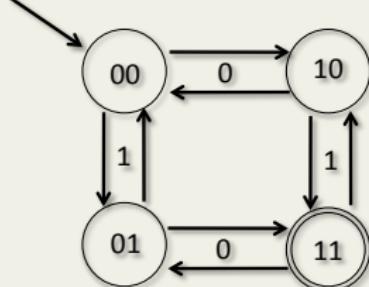
- A DFA M *accepts a string* w iff the unique path starting at the initial state and spelling out w ends at an accepting state.
- The *language accepted* (or “recognized”) by a DFA M is denoted $L(M)$ and defined by

$$L(M) = \{ w \mid M \text{ accepts } w\}$$

Warning

- “ M accepts language L ” **does not mean** simply that M accepts each string in L .
- “ M accepts language L ” **means**
 M accepts each string in L and no others!
- M “recognizes” L is a better term, but “accepts” is widely accepted (and recognized).

Examples: What is $L(M)$?

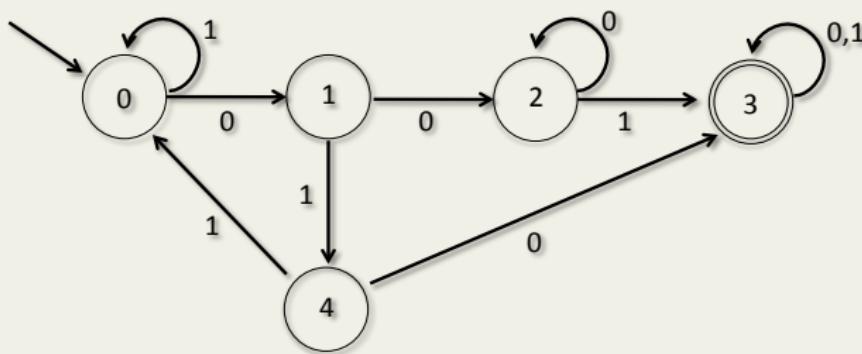


State = Memory

- The state of a DFA is its entire memory of what has come before
- The state must capture enough information to complete the computation on the suffix to come
- When designing a DFA, think “what do I need to know at this moment?” *That* is your state.

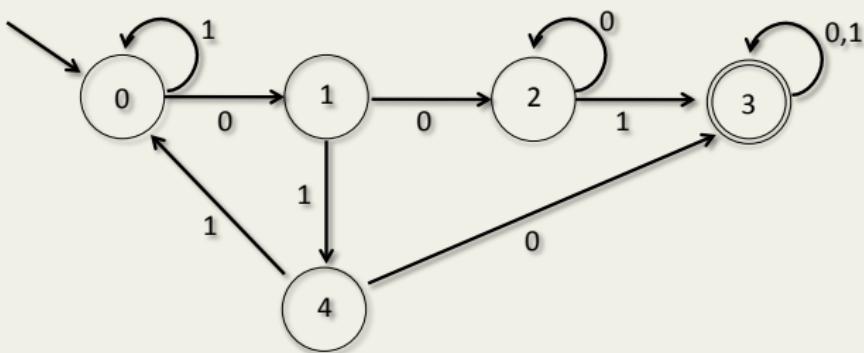
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



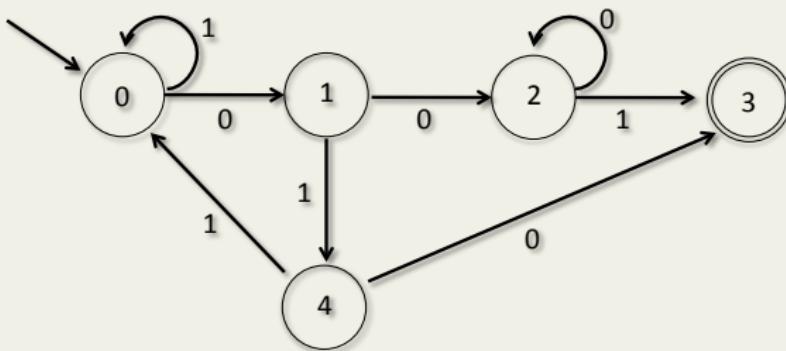
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



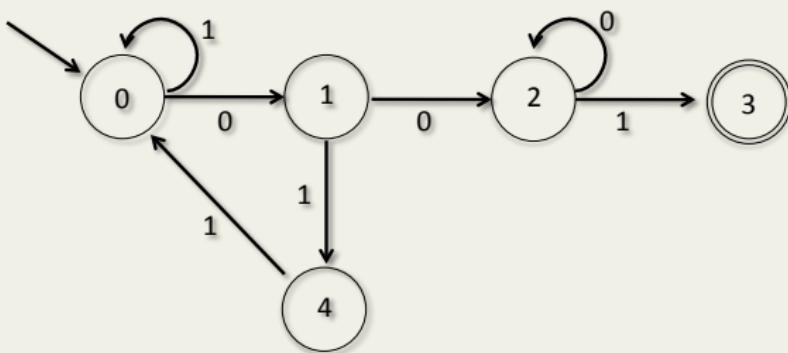
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



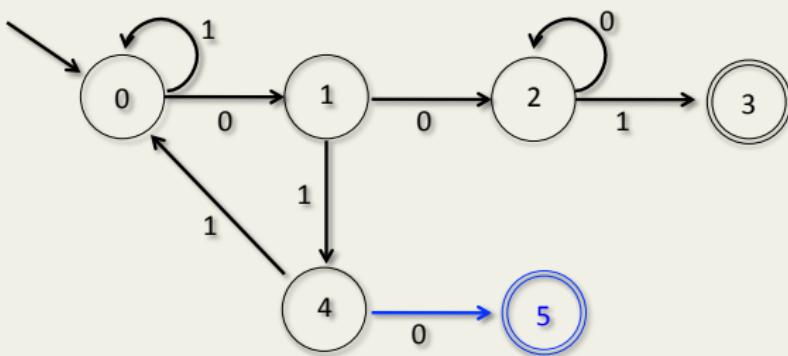
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



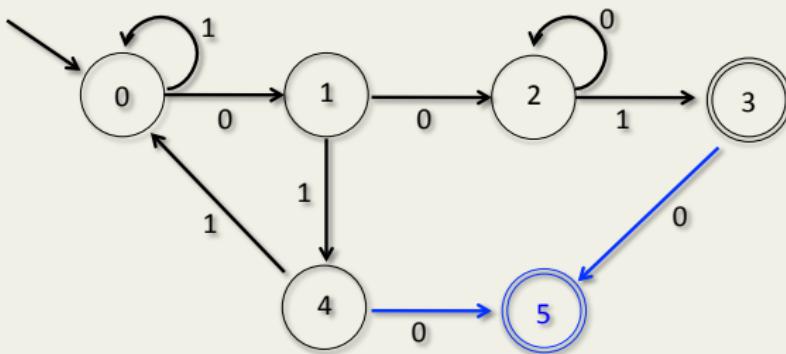
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



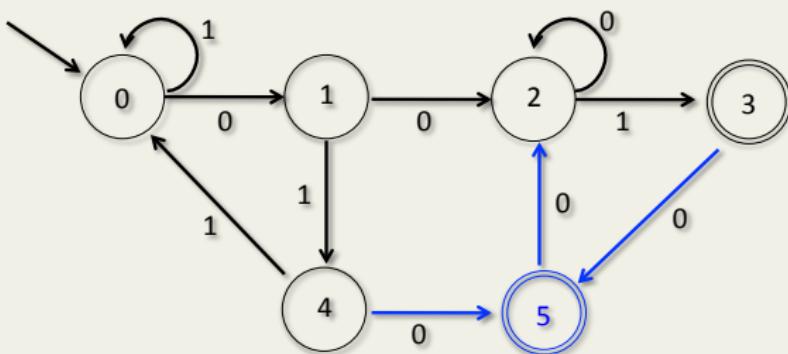
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



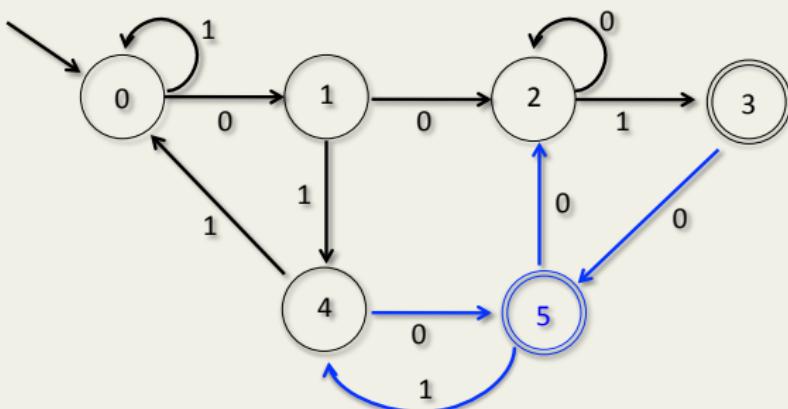
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



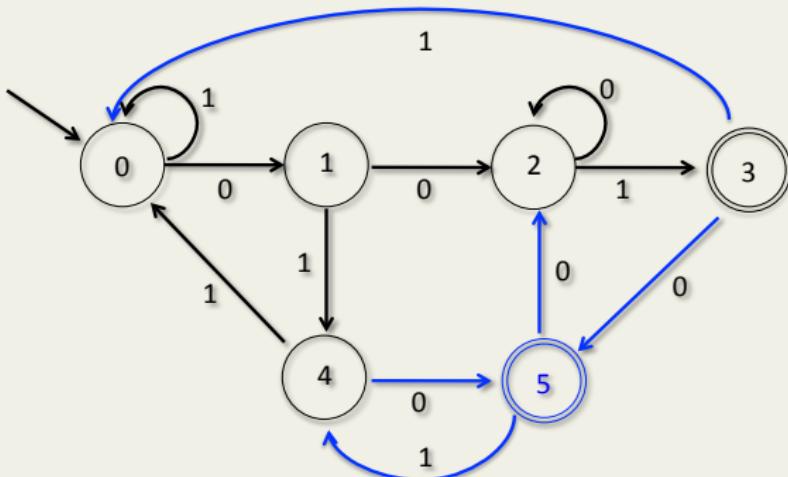
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



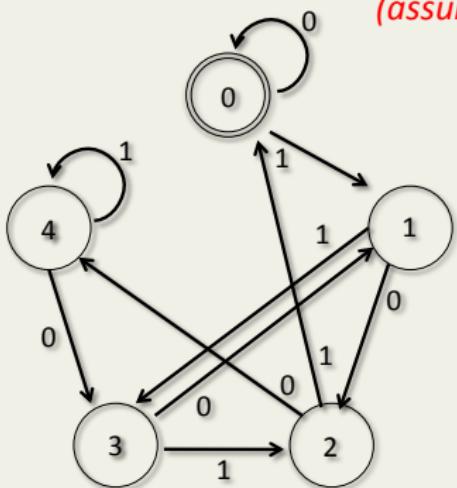
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$



Binary #s congruent to 0 mod 5

(assume no leading 0s)



Key Idea

If $w \text{ mod } 5 = a$, then:

- $w_0 \text{ mod } 5 = 2a \text{ mod } 5$
- $w_1 \text{ mod } 5 = 2a+1 \text{ mod } 5$

Test: $1101011 = 107 = 2 \text{ mod } 5$

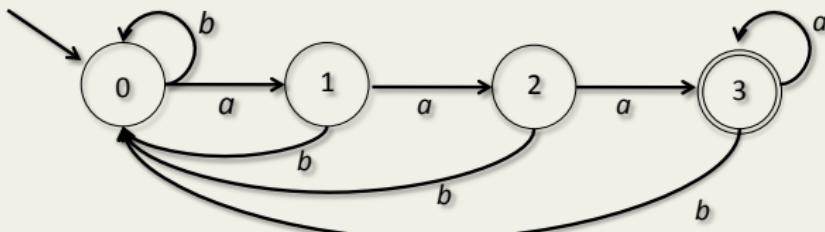
Formal (tuple) Representation

Sometimes, it is easier to specify the DFA using this formalism, instead of drawing a graph

A DFA is a quintuple $M=(Q,\Sigma,\delta,q_0,F)$, where:

- Q is a finite set of *states*
- Σ is a finite *alphabet* of symbols
- $\delta: Q \times \Sigma \rightarrow Q$ is a *transition function*
- q_0 is the *initial state*
- $F \subseteq Q$ is the set of *accepting states*

Example



- $Q = \{0,1,2,3\}$
- $\Sigma = \{a,b\}$
- δ specified at right
- $q_0 = 0$
- $F = \{3\}$

state \ input	a	b
0	1	0
1	2	0
2	3	0
3	3	0

Extending δ

- $\delta(q, a) = p$ means in graph that $q \xrightarrow{a} p$
- But how can we define $\delta(q, w)$ to express $q \xrightarrow{w} p$
- Must extend $\delta: Q \times \Sigma^* \rightarrow Q$
 - $\delta(q, \varepsilon) = q$ for every q ; $\delta(q, a)$ already defined
 - $\delta(q, au) = \delta(\delta(q, a), u)$ for $|u| \geq 1$, all q, a



take first step according to δ



take rest of steps inductively according to δ

$\delta(q, w) = p$ corresponds to $q \xrightarrow{w} p$

Formal definition of $L(M)$

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA

Then $L(M) = \{w \mid \delta(q_0 w) \in F\}$

We will show later that:

Theorem

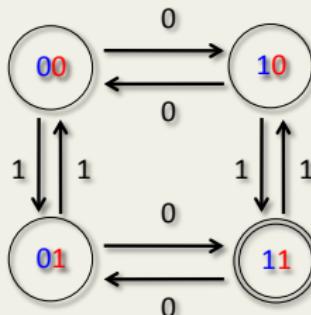
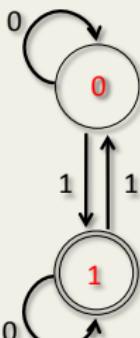
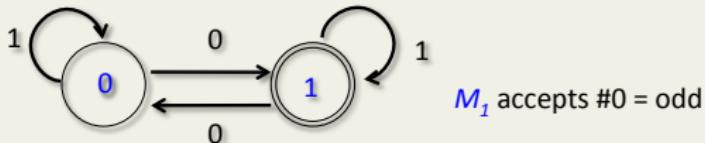
L is regular if and only if $L = L(M)$ for some DFA M

Example use

$L(M) = \{w \mid w \text{ in base } b \text{ is congruent to } k \bmod m\}$

- $Q = \{0, 1, \dots, m-1\}$
- $\Sigma = \{0, 1, \dots, b-1\}$
- $q_0 = 0$
- $\delta(n, a) = bn + a \bmod m$
- $F = \{k\}$

M simulating both M_1 and M_2



Cross-product machine

M accepting $L(M_1) \cap L(M_2)$

$$Q = Q_1 \times Q_2$$

$$q_0 = (q_0^{(1)}, q_0^{(2)})$$

$$F = F_1 \times F_2 = \{ (q_1, q_2) \mid q_1 \text{ in } F_1 \text{ and } q_2 \text{ in } F_2 \}$$

Transition function:

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

$(q_1, q_2) \xrightarrow{a} (p_1, p_2)$ if and only if

- $q_1 \xrightarrow[1]{a} p_1$ in M_1
- $q_2 \xrightarrow[2]{a} p_2$ in M_2

Proof that simulation is correct

- Induction on what? that what?
- Will need to prove that action of machine is correct starting from any states.
- We know that:

$$(q_1, q_2) \xrightarrow{a} (p_1, p_2) \text{ iff}$$

- $q_1 \xrightarrow[1]{a} p_1$ in M_1
- $q_2 \xrightarrow[2]{a} p_2$ in M_2

By definition

Show that:

$$(q_1, q_2) \xrightarrow{w} (p_1, p_2) \text{ iff}$$

- $q_1 \xrightarrow[1]{w} p_1$ in M_1
- $q_2 \xrightarrow[2]{w} p_2$ in M_2

Just like definition of δ ,
but with w instead of a

DEF for M:

$$(q_1, q_2) \xrightarrow{a} (p_1, p_2)$$

means

$$q_1 \xrightarrow{a_1} p_1 \text{ and } q_2 \xrightarrow{a_2} p_2$$

BY INDUCTION
on $|w|$

$$(q_1, q_2) \xrightarrow{w} (p_1, p_2)$$

iff

$$q_1 \xrightarrow{w_1} p_1 \text{ and } q_2 \xrightarrow{w_2} p_2$$

$$(q_1, q_2) \xrightarrow{a \mid u} (p_1, p_2)$$

$$(r_1, r_2)$$

pull apart the computation

$$(q_1, q_2) \xrightarrow{a} (r_1, r_2)$$

apply definition

$$q_1 \xrightarrow{a_1} r_1$$

and

$$q_2 \xrightarrow{a_2} r_2$$

$$(r_1, r_2) \xrightarrow{u} (p_1, p_2)$$

apply inductive hypothesis since $|u| < |w|$

$$r_1 \xrightarrow{u_1} p_1$$

$$r_2 \xrightarrow{u_2} p_2$$

$$q_1 \xrightarrow{w_1} p_1$$

$$q_2 \xrightarrow{w_2} p_2$$

paste the computations back together

Finishing up...

- We proved:

$$\begin{array}{c} (q_1, q_2) \xrightarrow{w} (p_1, p_2) \\ \text{iff} \\ q_1 \xrightarrow{w_1} p_1 \text{ and } q_2 \xrightarrow{w_2} p_2 \end{array}$$

By definition, w accepted by M

$$\text{iff } (q_0^{(1)}, q_0^{(2)}) \xrightarrow{w} (f_1, f_2) \text{ in } F_1 \times F_2$$

$$\text{iff } q_0^{(1)} \xrightarrow{w} f_1 \text{ in } F_1 \text{ AND } q_0^{(2)} \xrightarrow{w} f_2 \text{ in } F_2$$

$$\text{iff } w \text{ in } L(M_1) \text{ AND } w \text{ in } L(M_2)$$

Formal proof that simulation is correct

- We know by definition that:
 - for all q_1 in Q_1 , for all q_2 in Q_2
 - for all characters a
 $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- We prove by induction on $|w|$ that:
 - for all q_1 in Q_1 , for all q_2 in Q_2
 - for all strings w
 $\delta((q_1, q_2), w) = (\delta_1(q_1, w), \delta_2(q_2, w))$

Looks just like definition of δ , but with w instead of a

To prove: $\delta((q_1, q_2), w) = (\delta_1(q_1, w), \delta_2(q_2, w))$

Induction on $|w|$

- Base Case: $|w| = 0$, so $w = \varepsilon$.

$$\delta((q_1, q_2), \varepsilon) = (q_1, q_2) = (\delta_1(q_1, \varepsilon), \delta_2(q_2, \varepsilon))$$

- Assume true for strings u of length $< n$.
- Let $w = au$ be an arbitrary string of length n .
- $\delta((q_1, q_2), au)$

$$= \delta(\delta((q_1, q_2), a), u) \quad \text{defn of } \delta \text{ extension}$$

$$= \delta((\delta_1(q_1, a), \delta_2(q_2, a)), u) \quad \text{by defn of } \delta$$

$$= \delta((r_1, r_2), u) \quad \text{define } r's \text{ to simplify}$$

$$= (\delta_1(r_1, u), \delta_2(r_2, u)) \quad \text{by induction } (|u| < n)$$

$$= (\delta_1(\delta_1(q_1, a), u), \delta_2(\delta_2(q_2, a), u))) \quad \text{get rid of } r's$$

$$= (\delta_1(q_1, au), \delta_2(q_2, au)) \quad \text{unsplitting}$$

$$= (\delta_1(q_1, w), \delta_2(q_2, w))$$

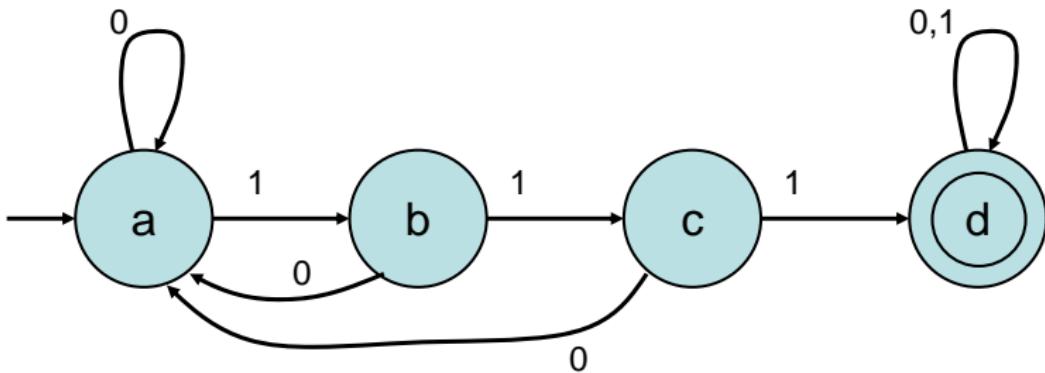
Properties of Regular languages

- We've shown how to accept intersection of two regular languages
- What about union?
- If L is accepted by a DFA, what about \overline{L} ?
- What about concatenation, and Kleene * ?
- Is there a DFA for $L_1 - L_2$ given M_1 and M_2 ?

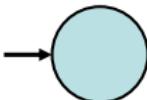
The answer to all of these questions, and more, is "Yes."

Example 1

- An FA diagram, machine M



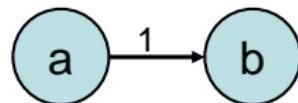
- Conventions:



Start state

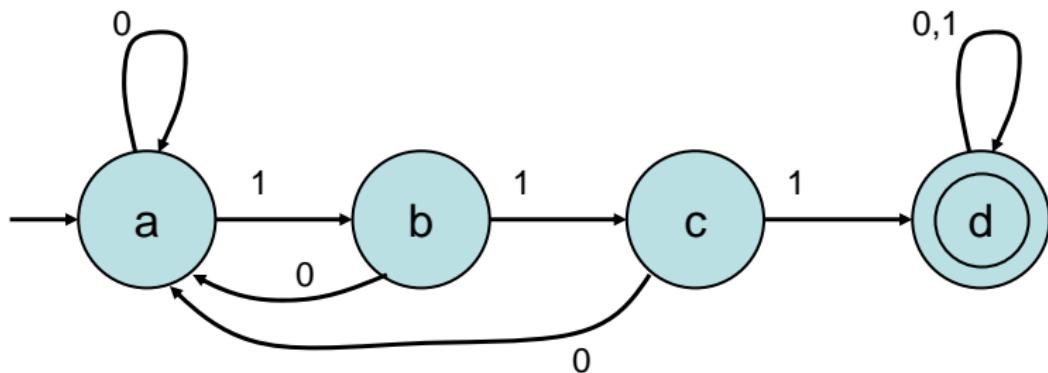


Accept state



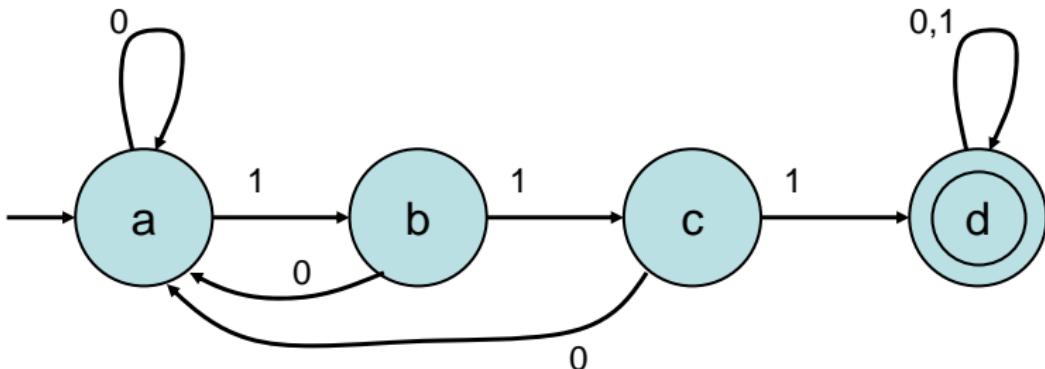
Transition from a to b on
input symbol 1.
Allow self-loops

Example 1



- Example computation:
 - Input word $w: 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0$
 - States: a b a b c a b c d d
- We say that **M accepts w**, since w leads to d, an accepting state.

Example 1



- What is $L(M)$ for Example 1?
- $\{ w \in \{0,1\}^* \mid w \text{ contains } 111 \text{ as a substring} \}$
- Note: Substring refers to consecutive symbols.

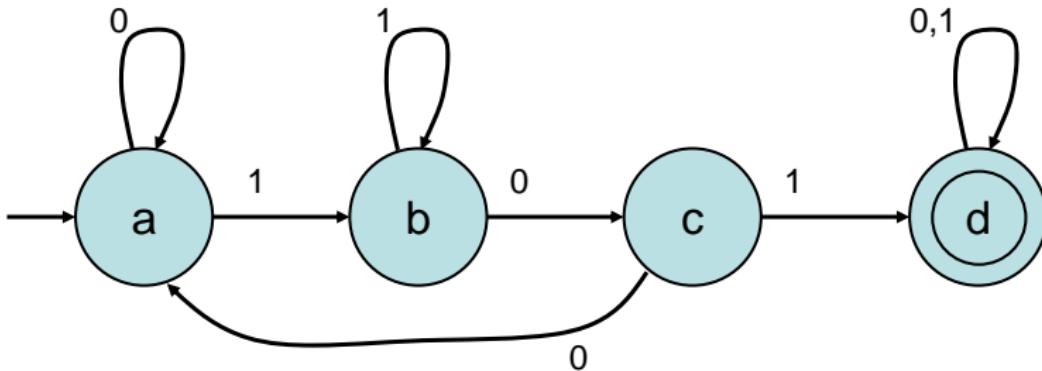
Example 1

- What is the 5-tuple $(Q, \Sigma, \delta, q_0, F)$?
- $Q = \{ a, b, c, d \}$
- $\Sigma = \{ 0, 1 \}$
- δ is given by the state diagram, or alternatively, by a table:
 - $q_0 = a$
 - $F = \{ d \}$

	0	1
a	a	b
b	a	c
c	a	d
d	d	d

Example 2

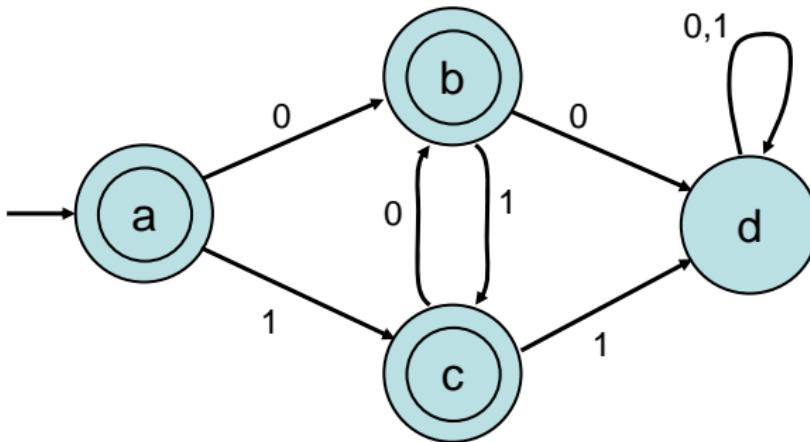
- Design an FA M with $L(M) = \{ w \in \{ 0,1 \}^* \mid w \text{ contains } 101 \text{ as a substring} \}$.



- Failure from state b causes the machine to remain in state b.

Example 3

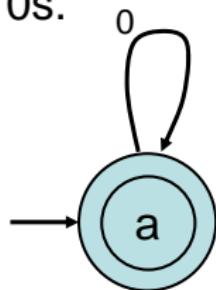
- $L = \{ w \in \{ 0,1 \}^* \mid w \text{ doesn't contain either } 00 \text{ or } 11 \text{ as a substring} \}$.



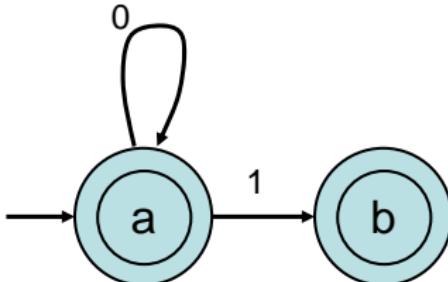
- State d is a **trap state** = a nonaccepting state that you can't leave.
- Sometimes we'll omit some arrows; by convention, they go to a trap state.

Example 4

- $L = \{ w \mid \text{all nonempty blocks of 1s in } w \text{ have odd length} \}$.
- E.g., ε , or 100111000011111, or any number of 0s.
- Initial 0s don't matter, so start with:



- Then 1 also leads to an accepting state, but it should be a different one, to "remember" that the string ends in one 1.

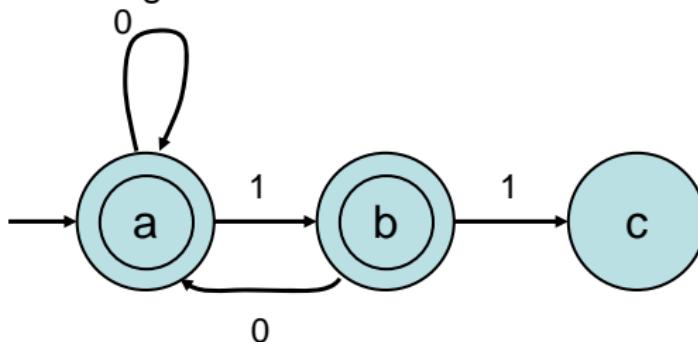
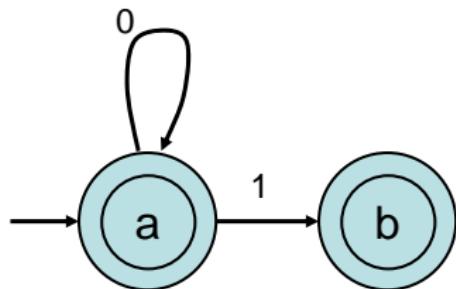


Example 4

- $L = \{ w \mid \text{all nonempty blocks of 1s in } w \text{ have odd length} \}$.

- From b:

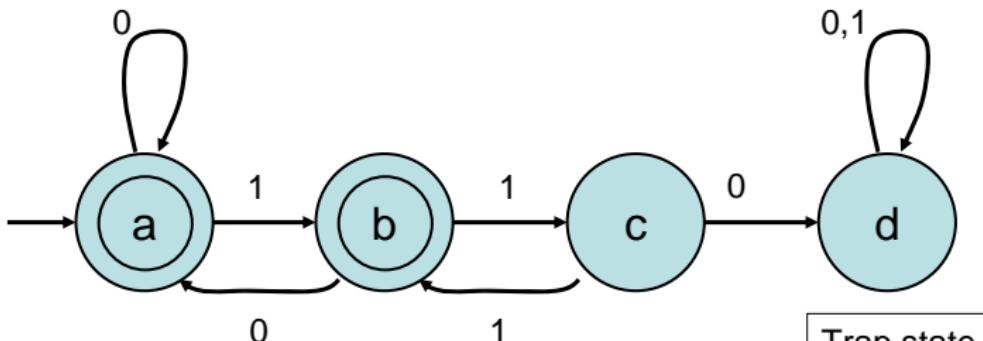
- 0 can return to a, which can represent either ϵ , or any string that is OK so far and ends with 0.
 - 1 should go to a new **nonaccepting state**, meaning “the string ends with two 1s”.



- Note: c isn't a trap state---we can accept some extensions.

Example 4

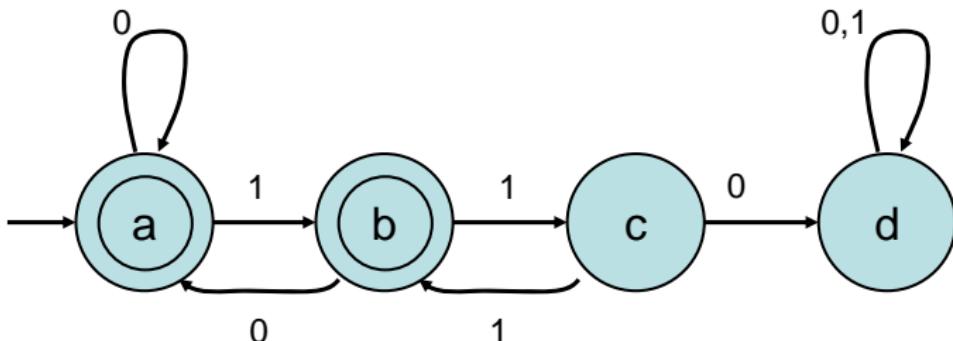
- $L = \{ w \mid \text{all nonempty blocks of 1s in } w \text{ have odd length} \}$.



- From c:
 - 1 can lead back to b, since future acceptance decisions are the same if the string so far ends with any odd number of 1s.
 - Reinterpret b as meaning “ends with an odd number of 1s”.
 - Reinterpret c as “ends with an even number of 1s”.
 - 0 means we must reject the current string and all extensions.

Example 4

- $L = \{ w \mid \text{all nonempty blocks of 1s in } w \text{ have odd length} \}$.



- Meanings of states (more precisely):
 - a: Either ϵ , or contains no **bad block** (even block of 1s followed by 0) so far and ends with 0.
 - b: No bad block so far, and ends with odd number of 1s.
 - c: No bad block so far, and ends with even number of 1s.
 - d: Contains a bad block.

Example 5

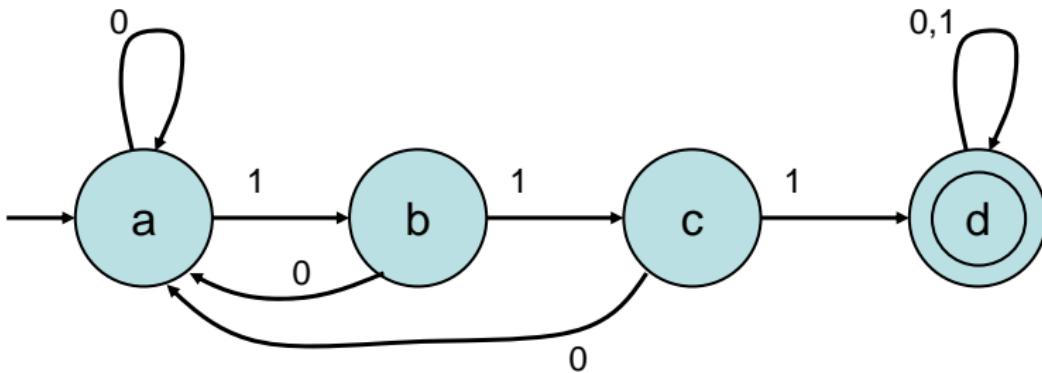
- $L = EQ = \{ w \mid w \text{ contains an equal number of } 0\text{s and } 1\text{s} \}$.
- No FA recognizes this language.
- Idea (not a proof):
 - Machine must “remember” how many 0s and 1s it has seen, or at least the difference between these numbers.
 - Since these numbers (and the difference) could be anything, there can’t be enough states to keep track.
 - So the machine will sometimes get confused and give a wrong answer.
- We’ll turn this into an actual proof next week.

Closure under operations

- The set of FA-recognizable languages is closed under all six operations (union, intersection, complement, set difference, concatenation, star).
- This means: If we start with FA-recognizable languages and apply any of these operations, we get another FA-recognizable language (for a different FA).
- **Theorem 1:** FA-recognizable languages are closed under complement.
- **Proof:**
 - Start with a language L_1 over alphabet Σ , recognized by some FA, M_1 .
 - Produce another FA, M_2 , with $L(M_2) = \Sigma^* - L(M_1)$.
 - Just interchange accepting and non-accepting states.

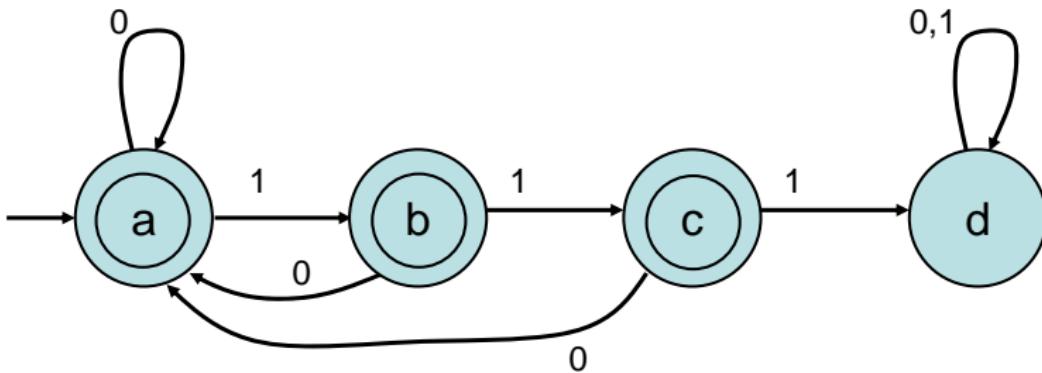
Closure under complement

- **Theorem 1:** FA-recognizable languages are closed under complement.
- **Proof:** Interchange accepting and non-accepting states.
- Example: FA for $\{ w \mid w \text{ does not contain } 111 \}$
 - Start with FA for $\{ w \mid w \text{ contains } 111 \}$:



Closure under complement

- **Theorem 1:** FA-recognizable languages are closed under complement.
- **Proof:** Interchange accepting and non-accepting states.
- Example: FA for $\{ w \mid w \text{ does not contain } 111 \}$
 - Interchange accepting and non-accepting states:



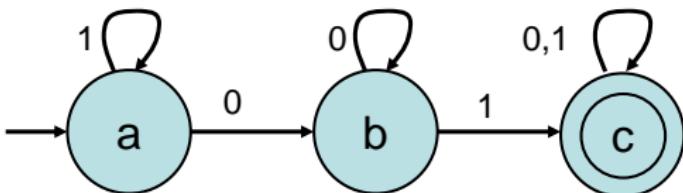
Closure under intersection

- **Theorem 2:** FA-recognizable languages are closed under intersection.
- **Proof:**
 - Start with FAs M_1 and M_2 for the same alphabet Σ .
 - Get another FA, M_3 , with $L(M_3) = L(M_1) \cap L(M_2)$.
 - Idea: Run M_1 and M_2 “in parallel” on the same input. If both reach accepting states, accept.
 - Example:
 - $L(M_1)$: Contains substring 01.
 - $L(M_2)$: Odd number of 1s.
 - $L(M_3)$: Contains 01 and has an odd number of 1s.

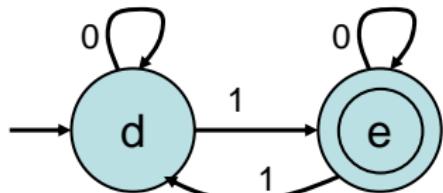
Closure under intersection

- Example:

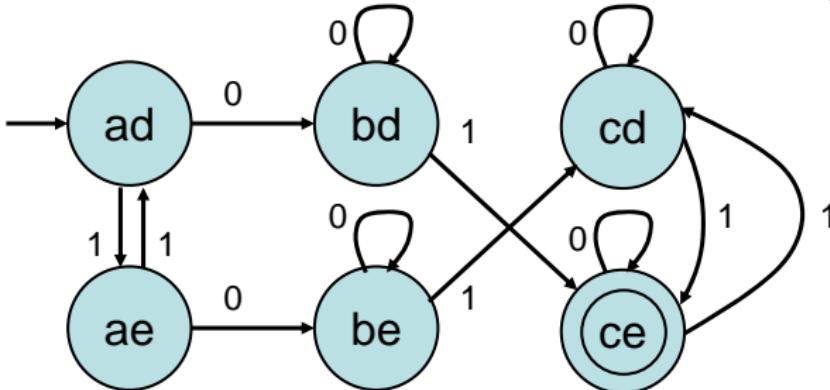
M_1 : Substring 01



M_2 : Odd number of 1s



M_3 :



Closure under intersection, general rule

- Assume:
 - $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$
 - $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
- Define $M_3 = (Q_3, \Sigma, \delta_3, q_{03}, F_3)$, where
 - $Q_3 = Q_1 \times Q_2$
 - Cartesian product, $\{(q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2\}$
 - $\delta_3((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
 - $q_{03} = (q_{01}, q_{02})$
 - $F_3 = F_1 \times F_2 = \{ (q_1, q_2) \mid q_1 \in F_1 \text{ and } q_2 \in F_2 \}$

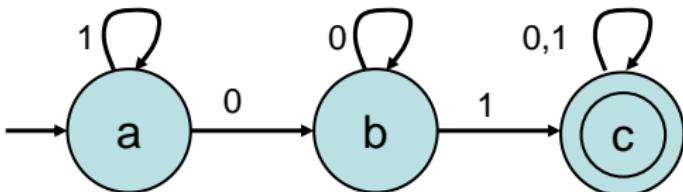
Closure under union

- **Theorem 3:** FA-recognizable languages are closed under union.
- **Proof:**
 - Similar to intersection.
 - Start with FAs M_1 and M_2 for the same alphabet Σ .
 - Get another FA, M_3 , with $L(M_3) = L(M_1) \cup L(M_2)$.
 - Idea: Run M_1 and M_2 “in parallel” on the same input. If either reaches an accepting state, accept.
 - Example:
 - $L(M_1)$: Contains substring 01.
 - $L(M_2)$: Odd number of 1s.
 - $L(M_3)$: Contains 01 or has an odd number of 1s.

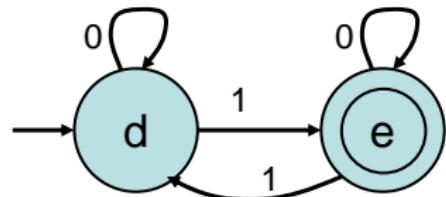
Closure under union

- Example:

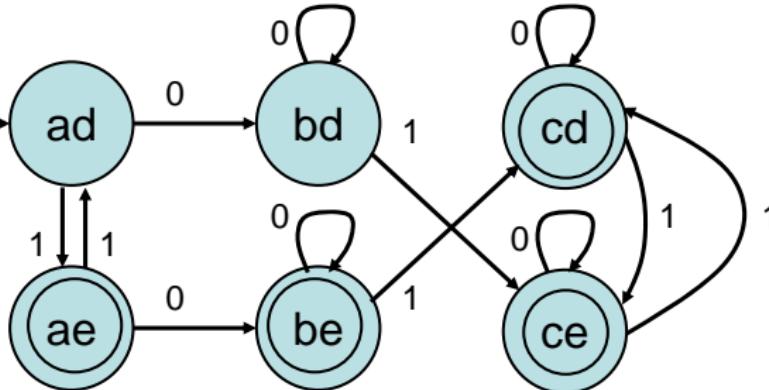
M_1 : Substring 01



M_2 : Odd number of 1s



M_3 : 1



Closure under union, general rule

- Assume:
 - $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$
 - $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
- Define $M_3 = (Q_3, \Sigma, \delta_3, q_{03}, F_3)$, where
 - $Q_3 = Q_1 \times Q_2$
 - Cartesian product, $\{(q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2\}$
 - $\delta_3((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
 - $q_{03} = (q_{01}, q_{02})$
 - $F_3 = \{ (q_1, q_2) \mid q_1 \in F_1 \text{ or } q_2 \in F_2 \}$

Closure under set difference

- **Theorem 4:** FA-recognizable languages are closed under set difference.
- **Proof:**
 - Similar proof to those for union and intersection.
 - Alternatively, since $L_1 - L_2$ is the same as $L_1 \cap (L_2)^c$, we can just apply Theorems 2 and 3.

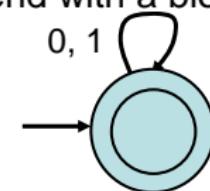
Closure under concatenation

- **Theorem 5:** FA-recognizable languages are closed under concatenation.
- **Proof:**
 - Start with FAs M_1 and M_2 for the same alphabet Σ .
 - Get another FA, M_3 , with $L(M_3) = L(M_1) \circ L(M_2)$, which is $\{ x_1 x_2 \mid x_1 \in L(M_1) \text{ and } x_2 \in L(M_2) \}$
 - Idea: ???
 - Attach accepting states of M_1 somehow to the start state of M_2 .
 - But we have to be careful, since we don't know when we're done with the part of the string in $L(M_1)$ ---the string could go through accepting states of M_1 several times.

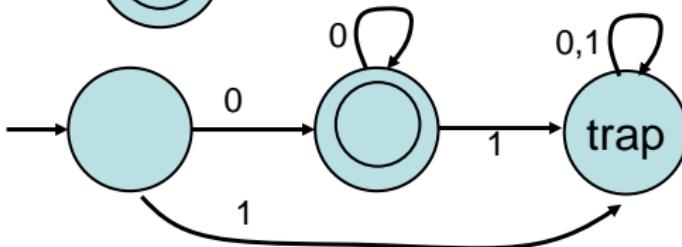
Closure under concatenation

- **Theorem 5:** FA-recognizable languages are closed under concatenation.
- **Example:**

- $\Sigma = \{0, 1\}$, $L_1 = \Sigma^*$, $L_2 = \{0\} \{0\}^*$ (just 0s, at least one).
- $L_1 L_2 =$ strings that end with a block of at least one 0
- M_1 :



- M_2 :



- How to combine?
- We seem to need to “guess” when to shift to M_2 .
- Leads to our next model, NFAs, which are FAs that can guess.

Closure under star

- **Theorem 6:** FA-recognizable languages are closed under star.
- **Proof:**
 - Start with FA M_1 .
 - Get another FA, M_2 , with $L(M_2) = L(M_1)^*$.
 - Same problems as for concatenation---need guessing.
 - ...
 - We'll define NFAs next, then return to complete the proofs of Theorems 5 and 6.