Finite Automata
(FAs)

- **Automata: The General Idea**
- **Diagrams and Language Recognition**
- **Formal Notation**
- **Nondeterministic Finite Automata**
- **Removing Nondeterminism**

Why FAs ?

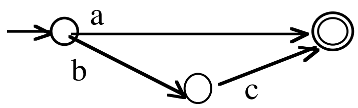
- ***Recognition*** **: Is string x in language L?**
- ***Simplest*** **way to specify some languages**
- ***Equivalences*** **among representation systems**
- **First in a family of *models* of *computation***

States:
the essence of FAs

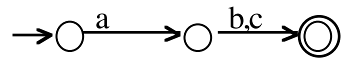
- ***State*** **: a model of the entire internal memory**
- **FA's *memory* : knowing what state it's in.**
- **State to state *transitions*, based on input**
- **State-transition *diagrams* (next)**

State Transition Diagrams for some languages

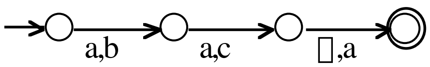
{a, bc}



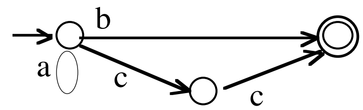
{ab, ac}



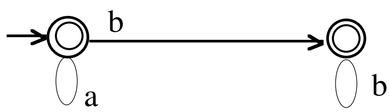
{aa,ac,ba,bc, aaa,aca,baa,bca}



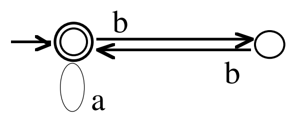
{b,cc,ab,acc,aab,aacc,…}



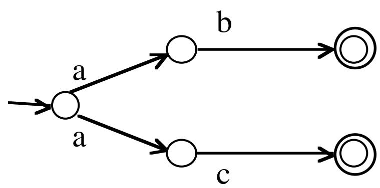
{□, a,b,aa,ab,bb,aaa,aab,abb,bbb,…}



{□, a,bb,aa,abb,bba,bbbb,aaa,…}



{ab, ac}



Reading a State Transition Diagram

What to do with the…	state:	string:
start at the beginning:	start state;	start of string
go right on to the end:	follow an arrow;	use up a symbol
then stop:	accepting state;	end of string

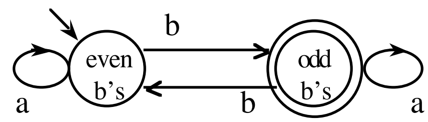
Conventions
for State Transition Diagrams

- Arrow with 2 labels: shorthand for 2 transitions.
- If the start state is an accepting state, □ is accepted.
- Stopping at an accepting state is optional.
- Rejection: if input ends at a non-accepting state.
- Rejection: if (state, input) pair has no arrow.

<u>From Description to Diagram</u>

Find a state-transition diagram for strings over Σ = {a,b} that have an odd number of b's.

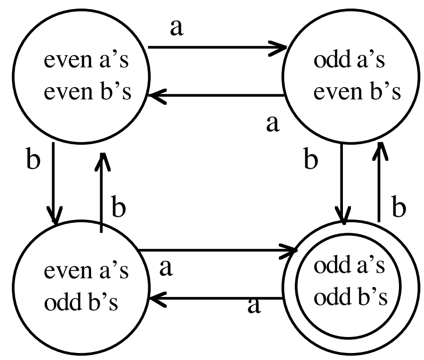
Using states to keep track of whether the number of b's so far is odd or even.



<u>**Description to Diagram**</u>

Find a state-transition diagram for those strings over □ = {a,b} that have both an odd number of a's *and* an odd number of b's.

Using states to refect even/odd combinations.



Formal Notation: The Quintuple

A finite automaton, M, is a quintuple, $(Q, \Sigma, q_0, \delta, A)$, where

- Q is a finite set of states,
- Σ is a finite set of symbols,
- $q_0 \in Q$ and q_0 is the start state,
- $\delta: Q \times S \rightarrow Q$,
- $A \subseteq Q$ and A is the set of accepting states.

Example of a Quintuple

Let $M = (Q, \Sigma, q_0, \delta, A)$

be the FA with

$Q = \{q_0\}$,

$\Sigma = \{a\}$,

$A = \{q_0\}$, and

$\delta(q_0, a) = q_0$.

- Draw the state-transition diagram.
- What is its language?

Formal Notation,
a Diagram
and a Trap state

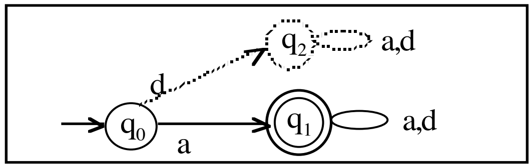
Let $M = (\{q_0, q_1, q_2\}, \{a, d\}, q_0, □, \{q_1\})$, …

where □ is specified by this table:

□	a	d
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_2	q_2

- **Which is the trap state?**
- **Draw the Diagram.**

The Diagram, with Trap State



- *Trap state* is shown dotted …
- … as are the transitions to and from it.
- What (else) tells you it's the trap state?
- What language is expressed?

The Language

- **The language of the diagram is** $\{a\}\ \{a,d\}^*$
- **What has this to do with programming languages?**

δ^*
is a Function

Let $M = (Q, \Sigma, q_0, \delta, A)$ be an FA.

- δ^* is a function.
- It maps a state and a string …
- to a resulting state:

$$\delta^* : Q \times \Sigma^* \to Q$$

Recursive Definition of δ^*

For any $q \in Q$,

$$\delta^*(q, \varepsilon) = q$$

For any $q \in Q$, any $a \in \Sigma$ and any $x \in \Sigma^*$,

$$\delta^*(q, xa) = \delta(\delta^*(q,x), a).$$

The Language of an Automaton

Given a finite automaton *M*, where

$$M = (Q, \Sigma, q_0, \delta, A) \ ,$$

***M*'s language, L(*M*), the strings it accepts:**

$$L(M) = \{ x \in \Sigma^* \mid \delta^*(q_0, x) \in A \}$$

***M* rejects the strings of (the complement) $\Sigma^* \setminus L(M)$.**

Recognizing the Language of an Automaton

Let *M* be the FA, (*Q*,□,*q*₀,*A*,□),

x the input string, and

"□ " the assignment operator.

Let *dequeue* remove and return the first symbol of a string.

q □ q_0
while x ≠ □ *do*
  □ □ *dequeue*(x)
  q □ □(q, □)
return q □ A

<u>NonDeterminism</u>

- **Definition: 2+ transitions/arrows**

  with the same input/label, and

  from the same state

- **Motivating Examples**

  All strings ending in abb, or

  Results of –removal (later)

Removing NonDeterminism: The Subset Construction

- Think of partial knowledge as a set of states.
- New machine: states are ‘states of mind.’
- So new state is a set of old states.
- New start state contains just the old one.
- New state accepts if it includes old accepters.

□-Elimination

- **For each □****a_i* **path from** *q_j* **to** *q_k***, add an** *a_i* **transition from** *q_j* **to** *q_k***.**

- **For each □* path from** *q_j* **to an accepting state** *q_k***, make** *q_j* **accepting.**

- **Remove all □-transitions.**

Tidying Up after □-Elimination

- **No Unreachables:** **Remove unreachable states and transitions from them.**

- **Careful Merging:** **A set of states can be merged if they agree about <u>acceptance</u> and <u>destination</u>**

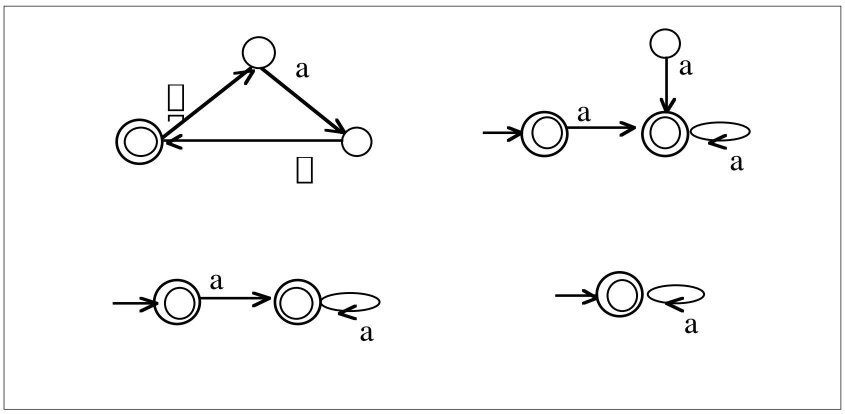
  - ✓ **all or none of them are accepting and**

  - ✓ **all would have the same transition destination, – if any – for each label, after completion of the merger.**

Example Using 3 Simplifications

Each diagram accepts the language of {*a*}*

- ✓ **The first one is from {*a*}* in Figure 10.2.**
- ✓ **□-elimination yields the second.**
- ✓ **Eliminate an unreachable state to get the second.**
- ✓ **Then carefully merge to get the last one.**



<u>Equivalences we have shown</u>

- **the languages from 3 language ops**
- **… are within the capacity of DFA diagrams**
- **… and of NFA-□s**
- **… and NFAs (without □s)**
- **…and DFAs**