

# CSCI 256: Theory of Computing

## CHAPTER 2 *Finite Automata*

**Dr. Benjoe A. Juliano**

Tel. 530 898-4619 (office)  
530 898-6442 (dept)  
Fax. 530 898-5995

Juliano@ecst.csuchico.edu  
http://www.ecst.csuchico.edu/~juliano



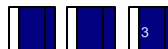
# Finite Automata: Informal Picture

- **Example: An “Electronic money” protocol**
- Participants:
  - Customer
  - Store
  - Bank
- Ground Rules:
  - The customer may decide to *pay*; that is, send money to the store.
  - The customer may decide to *cancel*; that is, the money is sent to the bank with the message that its value is added to the customer’s bank account.



# Finite Automata: Informal Picture

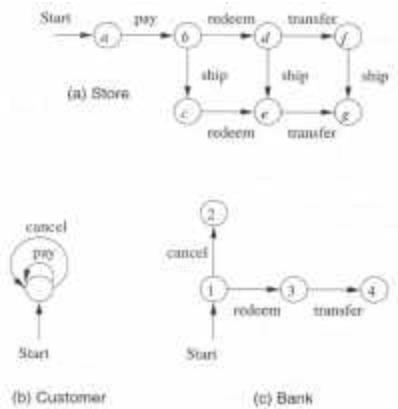
- **Example: An “Electronic money” protocol**
- Ground Rules (*continued*):
  - The store may *ship* goods to the customer.
  - The store may *redeem* the money; that is, the money is sent back to the bank with a request that its value be added to the store’s bank account.
  - The bank may *transfer* the money by creating a new, suitably encrypted money file and sending it to the store.
- Appropriate behavior of the three participants embodied in a protocol that could be represented as a finite automata ...



# Finite Automata: Informal Picture

- **Example: An “Electronic money” protocol**

- Finite automata representing participants:



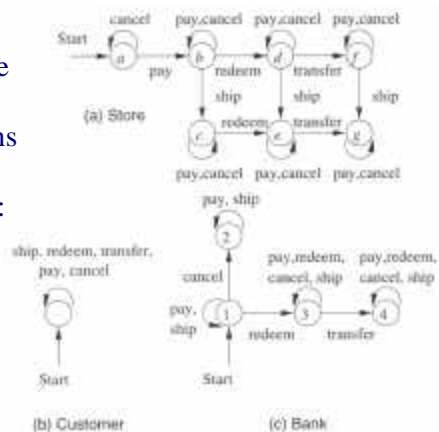
From Figure 2.1 of IATC - Hopcroft, Motwani, & Ullman, 2001.



## Finite Automata: Informal Picture

- Example: An “Electronic money” protocol

- Complete sets of transitions for the automata:



## Finite Automata: Informal Picture

- Example: An “Electronic money” protocol
- Validating the protocol via the product automaton:
  - Consider state (3,e): goods have been *ship*ped, but there will eventually be a transition to state (4,g) when the bank receives a *transfer* message. Elaborate ...
  - Consider state (2,c): bank receives a *cancel* message before a *redeem* message. This is an accessible state, but the only arc out leads back to that state. Elaborate ...

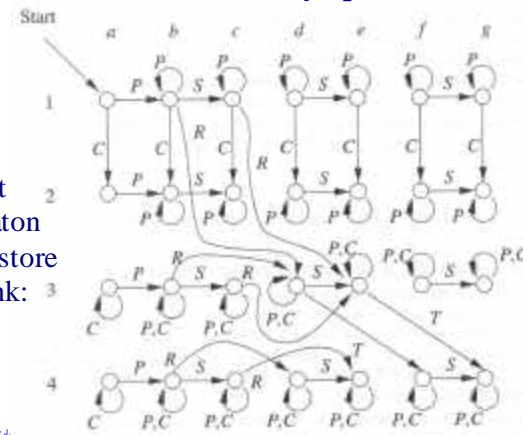
B. Juliano © 2002, based on notes by J. Ullman

7

## Finite Automata: Informal Picture

- Example: An “Electronic money” protocol

- Product automaton for the store and bank:



B. Juliano © 2002, based on notes by J.

## Finite Automata

- Finite automata
  - an important way to describe certain simple, but highly useful languages called “regular languages.”
  - A graph with a finite number of nodes called *states* (typically  $Q$ ).
  - Arcs are labeled with one or more symbols from some *alphabet* (typically  $\Sigma$ ).
  - One state is designated as the *start state* or *initial state* (typically  $q_0$ ); some states are *final states* or *accepting states* (typically  $F$ ).

B. Juliano © 2002, based on notes by J. Ullman

8

## Finite Automata

- **Finite automata**
  - A *transition function* (typically  $\delta$ )
    - that takes a state and input symbol as arguments
    - that returns a state
    - where each “rule” of  $\delta$  would be written as  $\delta(q,a)=p$ , where  $q,p \in Q$  and  $a \in \Sigma$ .
    - intuitively, if a FA is in state  $q \in Q$ , and input  $a \in \Sigma$  is received, then the FA goes to state  $p \in Q$  (it is not necessarily the case that  $q \neq p$ ).

B. Juliano © 2002, based on notes by J. Ullman

43

## Finite Automata

- A FA is represented as the five-tuple
 
$$A = (Q, S, \delta, q_0, F)$$
- Conventions:
  - Input symbols typically  $a, b$ , etc., or digits.
  - Strings of input symbols typically  $u, v, \dots, z$ .
  - States typically  $q, p$ , etc.

B. Juliano © 2002, based on notes by J. Ullman

43

## Deterministic Finite Automata

- **Definition**
  - A *deterministic finite automaton* consists of
    - A finite set of *states*, often denoted  $Q$
    - A finite set of *input symbols*, often denoted  $S$
    - A *transition function*  $\delta: Q \times S \rightarrow Q$
    - A *start state*  $q_0 \in Q$
    - A set of *final or accepting states*  $F \subseteq Q$
  - A DFA is also referred to using the five-tuple notation:

$$A = (Q, S, \delta, q_0, F)$$

B. Juliano © 2002, based on notes by J. Ullman

43

## Example: Clamping Logic

- **Notes:**
  - We may think of an *accepting state* as representing a “1” output and nonaccepting states as representing a “0” output.
  - A “clamping” circuit waits for a “1” input, and forever after makes a “1” output. However, to avoid clamping on spurious noise, we’ll design a FA that waits for two 1’s in a row, and “clamps” only then.

B. Juliano © 2002, based on notes by J. Ullman

43

## Example: Clamping Logic

- In general, we can think of a *state* as representing a summary of the history of what has been seen on the input so far. So we need:
  - state  $q_0$ , the *start state*, says that the most recent input (if there was one) was not a **1**, and we have never seen two **1**'s in a row.
  - state  $q_1$ , indicates we have never seen **11**, but the previous input was **1**.
  - state  $q_2$  is the only *accepting state*, indicating that we have at some time seen **11**.

B. Juliano © 2002, based on notes by J. Ullman

43

## Example: Clamping Logic

- Thus, DFAA =  $(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$  where  $\delta$  is given by:

$\delta$	0	1
$\textcircled{R}$ $q_0$	$q_0$	$q_1$
$q_1$	$q_0$	$q_2$
* $q_2$	$q_2$	$q_2$

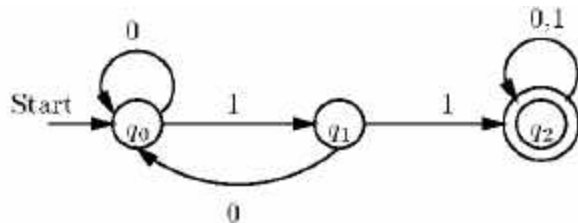
- by marking the start state with  $\textcircled{R}$  and accepting states with \*, the *transition table* that defines  $\delta$  also specifies the entire DFA.

B. Juliano © 2002, based on notes by J. Ullman

43

## Example: Clamping Logic

- The *transition diagram* depicting the DFAA is given by:



B. Juliano © 2002, based on notes by J. Ullman

43

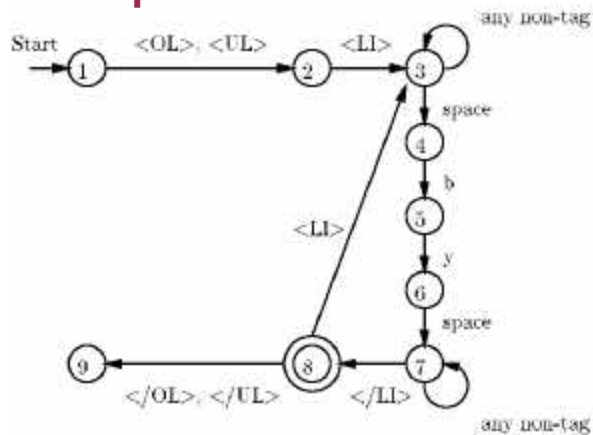
## Example: HTML documents

- Example:**
  - On the next slide is a DFA that scans HTML documents, looking for a list of what could be title-author pairs, perhaps in a reading list for some literature course.
  - The DFA *accepts* whenever it finds the end of a list item.

B. Juliano © 2002, based on notes by J. Ullman

43

## Example: HTML documents



B. Juliano © 2002, based on notes by J. Ullman

43

## Deterministic Finite Automata

### Extending $\delta$ to Strings

- If  $\delta$  is a transition function for a DFA, the **extended transition function** constructed from  $\delta$ , called  $\Delta$ , describes what happens when we start in any state and follow any sequence of inputs.
- By definition, for DFA  $A = (Q, \Sigma, \delta, q_0, F)$ 
  - $\Delta(q, \epsilon) = q$
  - For  $w \in \Sigma^*$ , if  $w = xa$  where  $a \in \Sigma$ , then  $\Delta(q, w) = \delta(\Delta(q, x), a)$

B. Juliano © 2002, based on notes by J. Ullman

43

## Deterministic Finite Automata

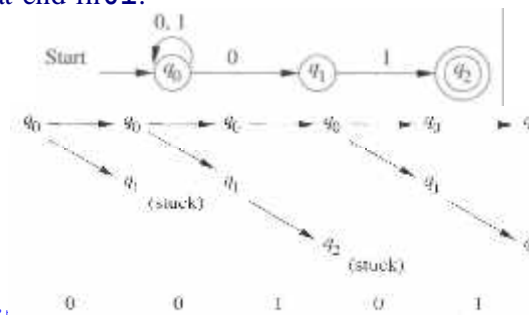
- The Language of a DFA**
- The **language** of a DFA  $A = (Q, \Sigma, \delta, q_0, F)$ , denoted  $L(A)$ , is defined by
 
$$L(A) = \{ w \mid \Delta(q_0, w) \in F \}$$
  - That is, the language of  $A$  is the set of strings  $w \in \Sigma^*$  that take the start state  $q_0$  to one of the accepting states.
  - If  $L$  is  $L(A)$  for some DFA  $A$ , then we say that  $L$  is a **regular language**.

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

- A **nondeterministic finite automata** is similar to a DFA, except for  $\delta: Q \times \Sigma \rightarrow 2^Q$
- NFA that accepts all and only the strings of  $\{0,1\}^*$  that end in  $01$ :



B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

- **Definition**
- A *nondeterministic finite automaton* consists of:
  - A finite set of *states*, often denoted  $Q$
  - A finite set of *input symbols*, often denoted  $S$
  - A *start state*  $q_0 \in Q$
  - A set of *final or accepting states*  $F \subseteq Q$
  - A *transition function*  $\delta: Q \times S \rightarrow 2^Q$
- A NFA is also referred to using the five-tuple notation:

$$A = (Q, S, \delta, q_0, F)$$

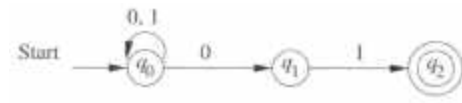
B. Juliano © 2002, based on notes by J. Ullman

43

From Figure 2.9 of IA TLC, Hopcroft, Motwani, &amp; Ullman, 2001.

## Nondeterministic Finite Automata

- **Representations**



$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

$\delta$	0	1
$q_0$	$\{q_0, q_1\}$	$\{q_0\}$
$q_1$	$\emptyset$	$\{q_2\}$
$q_2$	$\emptyset$	$\emptyset$

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

- **Extending  $\delta$  to Strings**
  - By definition, for  $NFA A = (Q, \Sigma, \delta, q_0, F)$ 
    - $\Delta(q, \epsilon) = \{q\}$
    - For  $w \in \Sigma^*$ , if
      - $w = xa$  where  $a \in \Sigma$ ; and
      - $\Delta(q, x) = \{p_1, p_2, \dots, p_k\}$   
 where  $\bigcup_i \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$
- then  $\Delta(q, w) = \{r_1, r_2, \dots, r_m\}$

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

- **The Language of an NFA**
- The *language* of a NFA  $A = (Q, \Sigma, \delta, q_0, F)$ , denoted  $L(A)$ , is defined by
 
$$L(A) = \{w \mid \Delta(q_0, w) \cap F \neq \emptyset\}$$
  - That is, the language of  $A$  is the set of strings  $w \in \Sigma^*$  such that  $\Delta(q_0, w)$  contains at least one accepting state.

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

- **Equivalence of DFAs and NFAs**
  - Every language that can be described by some NFA can also be described by some DFA.
    - but the DFA can have exponentially many states
  - Can be proven via **subset construction** ...
    - Start from NFA  $N=(Q_N, \Sigma, \delta_N, q_0, F_N)$
    - Goal: describe DFA  $D=(Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$  such that  $L(D)=L(N)$ .

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

- Given NFA  $N=(Q_N, \Sigma, \delta_N, q_0, F_N)$ , construct DFA  $D=(Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$  as follows:

- $Q_D = 2^{Q_N}$  (inaccessible states can be thrown)

- $F_D = \{S \in Q_D \mid S \cap F_N \neq \emptyset\}$

- For  $S \in Q_D$  and  $a \in \Sigma$ ,

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

$$\delta_D(\{q_1, q_2, \dots, q_k\}, a) =$$

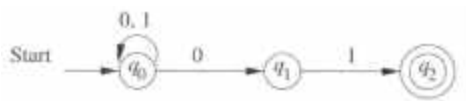
$$\delta_N(p_1, a) \cup \delta_N(p_2, a) \cup \dots \cup \delta_N(p_k, a)$$

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

- **Example**



		D		$\delta$		
				$\emptyset$	0	1
N	$\delta$	0	1	$\emptyset$	0	1
$\textcircled{0}$	$q_0$	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$q_1$	$\emptyset$	$\{q_2\}$	$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*$	$q_2$	$\emptyset$	$\emptyset$	$\{q_2\}$	$\emptyset$	$\emptyset$
				$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
				$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
				$\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
				$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

### Theorem 2.11

- If  $D=(Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$  is a DFA constructed from NFA  $N=(Q_N, \Sigma, \delta_N, q_0, F_N)$  by the *subset construction*, then  $L(D) = L(N)$ .
  - Proof (by induction on  $|w|$ ): *See textbook ...*

### Theorem 2.12

- A language  $L$  is accepted by some DFA if and only if  $L$  is accepted by some NFA.
  - Proof: *See textbook ...*

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

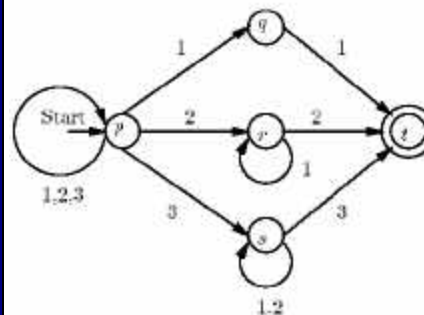
- **Example**
  - Design an NFA to accept strings over  $\Sigma = \{1, 2, 3\}$  such that the last symbol appears previously, without any intervening higher symbol; e.g.,  $w = 11, w = 21112, w = 312123$ .
    - Use start state to mean “I guess I have not seen the symbol that matches the ending symbol”
    - Use three other states to represent a guess that the matching symbol has been seen, and remembers what symbol that is.

B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

- **Example** *continued...*



	1	2	3
— p	pq	pr	ps
pq	pqt	pr	ps
*pq1	pqt	pr	ps
pr	pqr	prt	ps
*pr1	pqr	prt	ps
ps	pqs	prs	pst
*ps1	pqs	prs	pst
prs	prrs	prst	pst
*prst	prrs	prst	pst
pqs	pqst	prs	pst
*pqst	pqst	prs	pst
pqr	pqrl	prt	ps
*pqr1	pqrl	prt	ps
prrs	prrst	prst	pst
*prrst	prrst	prst	pst

- **Note:** Only 15 of the 32 possible states are accessible.

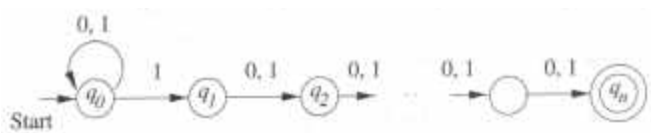
B. Juliano © 2002, based on notes by J. Ullman

43

## Nondeterministic Finite Automata

From Figure 2.15 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

- **A Bad Case for the Subset Construction**
  - Recall: Given an NFA, the equivalent DFA can have exponentially many states. *Why?*
  - Consider the NFA  $N$  where  $L(N)$  is the set of all strings in  $\{0, 1\}$  such that the  $n^{\text{th}}$  symbol from the end is 1.



**Note:** This NFA has no equivalent DFA with fewer than  $2^n$  states.

B. Juliano © 2002, based on notes by J. Ullman

43

## An Application: Text Search

- **Finding Strings in Text**
  - Given a set of words, find all documents that contain one (or all) of those words.
  - Inverted indexes
    - for each word, maintain a list of all the places where that word occurs.
  - Automaton-based techniques
    - Suitable for:
      - Dynamic, rapidly changing search repositories
      - news analyst, financial analyst
      - “shopping robot” searching for current prices
    - Documents cannot be cataloged

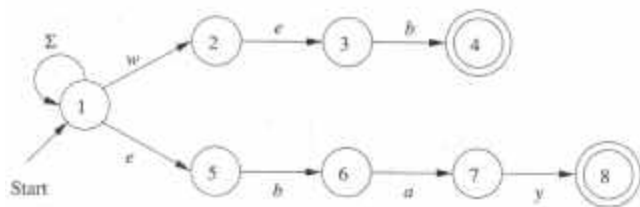
B. Juliano © 2002, based on notes by J. Ullman

43

From Figure 2.16 of *IA/TLC*, Hopcroft, Motwani, & Ullman, 2001.

## An Application: Text Search

- NFA for Text Search



An NFA that searches for the words "web" and "ebay".

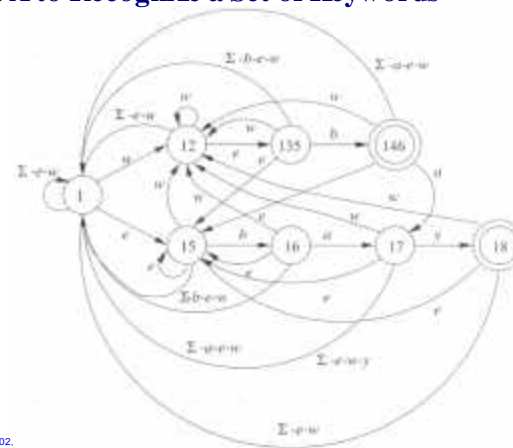
B. Juliano © 2002, based on notes by J. Ullman

43

From Figure 2.17 of *IA/TLC*, Hopcroft, Motwani, & Ullman, 2001.

## An Application: Text Search

- DFA to Recognize a Set of Keywords



B. Juliano © 2002.

43

## An Application: Text Search

- Automaton-based string matching algorithms

- Knuth-Morris-Pratt algorithm**

- Donald E. Knuth, James H. Morris, Jr., and Vaughn R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, 6(2): 323-350, June, 1977.

- Boyer-Moore algorithm**

- Robert S. Boyer and J. Strother Moore; "A fast string searching algorithm," *Communications of the ACM*, 20(10): 762-772, October, 1977.
  - Both algorithms have sub-algorithms that build the automata for recognizing the target text patterns.

B. Juliano © 2002, based on notes by J. Ullman

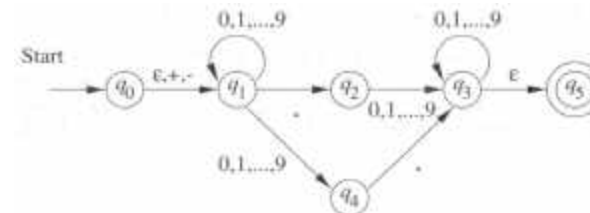
43

From Figure 2.18 of *IA/TLC*, Hopcroft, Motwani, & Ullman, 2001.

## Finite Automata with $\epsilon$ -transitions

- Uses of  $\epsilon$ -transitions

- Allow  $\epsilon$  to be a label on arcs
  - Example:



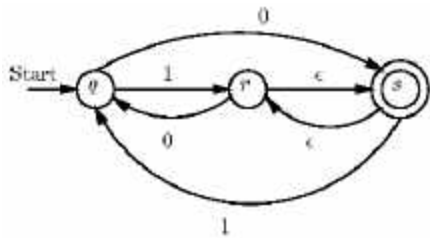
An  $\epsilon$ -NFA accepting decimal numbers

B. Juliano © 2002, based on notes by J. Ullman

43

## Finite Automata with $\epsilon$ -transitions

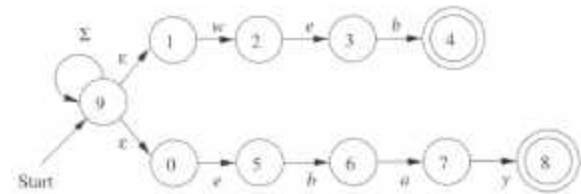
- Additional examples:



001 is accepted by the  $\epsilon$ -NFA above by the path  $q, s, r, q, r, s$ , with label  $001\epsilon = 001$ .

## Finite Automata with $\epsilon$ -transitions

- Additional examples:



Using  $\epsilon$ -transitions to help recognize keywords .

## Finite Automata with $\epsilon$ -transitions

- Formal Notation for an  $\epsilon$ -NFA**
  - An  $\epsilon$ -NFA consists of
    - A finite set of *states*, often denoted  $Q$
    - A finite set of *input symbols*, often denoted  $\Sigma$
    - A *start state*  $q_0 \in Q$
    - A set of *final or accepting states*  $F \subseteq Q$
    - A *transition function*  $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$
  - An  $\epsilon$ -NFA is also referred to using the five-tuple notation:

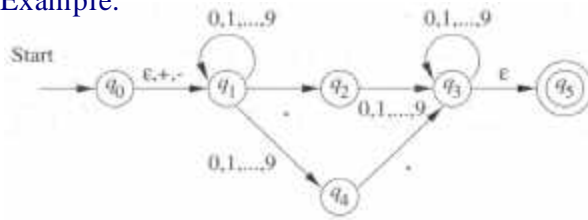
$$E = (Q, \Sigma, \delta, q_0, F)$$

## Finite Automata with $\epsilon$ -transitions

- $\epsilon$ -Closures**
  - Given a state  $q \in Q$  of  $\epsilon$ -NFA  $E = (Q, \Sigma, \delta, q_0, F)$ , the  **$\epsilon$ -closure of  $q$** , denoted  $\text{ECLOSE}(q)$ , is defined recursively as:
    - $q \in \text{ECLOSE}(q)$
    - If  $p \in \text{ECLOSE}(q)$  and  $\delta(p, \epsilon) = r$ , then  $r \in \text{ECLOSE}(q)$

## Finite Automata with $\epsilon$ -transitions

- **$\epsilon$ -Closures**
- Example:



An  $\epsilon$ -NFA accepting decimal numbers .

$$\text{ECLOSE}(q_0) = \{q_0, q_1\}$$

$$\text{ECLOSE}(q_3) = \{q_3, q_5\}$$

B. Juliano © 2002, based on notes by J. Ullman

43

## Finite Automata with $\epsilon$ -transitions

- **Extended Transition Function,  $\Delta$ , for  $\epsilon$ -NFAs**
  - By definition, for an  $\epsilon$ -NFA  $E = (Q, \Sigma, \delta, q_0, F)$ 
    - $\Delta(q, \epsilon) = \text{ECLOSE}(q)$
    - For  $w = xa$  where  $a \in \Sigma$ , to compute  $\Delta(q, w)$ :
      - Let  $\Delta(q, x) = \{p_1, p_2, \dots, p_k\}$
      - Let  $\bigcup_{1 \leq i \leq k} \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$
      - Then  $\Delta(q, w) = \bigcup_{1 \leq j \leq m} \text{ECLOSE}(r_j)$

B. Juliano © 2002, based on notes by J. Ullman

43

## Finite Automata with $\epsilon$ -transitions

- **Language of an  $\epsilon$ -NFA**
  - By definition, for an  $\epsilon$ -NFA  $E = (Q, \Sigma, \delta, q_0, F)$ , the **language** of  $E$ , denoted  $L(E)$ , is defined as

$$L(E) = \{ w \in \Sigma^* \mid \Delta(q_0, w) \cap F \neq \emptyset \}$$

B. Juliano © 2002, based on notes by J. Ullman

43

## Finite Automata with $\epsilon$ -transitions

- **Eliminating  $\epsilon$ -transitions**
  - Why?  $\epsilon$ -transitions are a convenience, but do not increase the power of FAs ...
  - Let  $\epsilon$ -NFA  $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ , then the equivalent DFA  $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$  is defined as follows:
    - $Q_D = 2^{Q_E}$
    - $q_D = \text{ECLOSE}(q_0)$
    - $F_D = \{ S \mid S \in Q_D \text{ and } S \cap F_E \neq \emptyset \}$

B. Juliano © 2002, based on notes by J. Ullman

43

## Finite Automata with $\epsilon$ -transitions

- **Eliminating  $\epsilon$ -transitions** *continued...*
  - For all  $a \in \Sigma$  and  $S \in Q_D$ , compute  $\delta_D(S, a)$  by
    - Let  $S = \{p_1, p_2, \dots, p_k\}$
    - Let  $\bigcup_{1 \leq i \leq k} \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$
    - Then  $\delta_D(S, a) = \text{ECLOSE}(\bigcup_{1 \leq i \leq k} \delta(p_i, a))$

### Theorem 2.22

- A language  $L$  is accepted by some  $\epsilon$ -NFA if and only if  $L$  is accepted by some DFA.

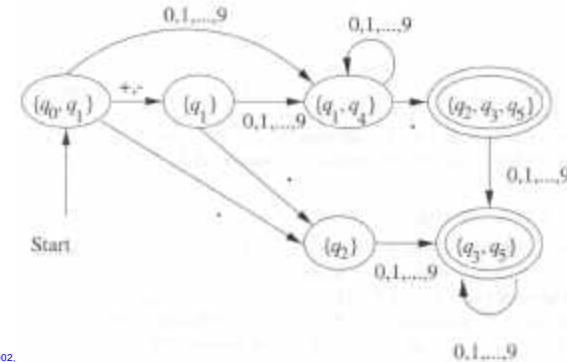
B. Juliano © 2002, based on notes by J. Ullman

43

From Figure 2.22 of JALC, Hopcroft, Motwani, &amp; Ullman, 2001.

## Finite Automata with $\epsilon$ -transitions

- **Eliminating  $\epsilon$ -transitions** *continued...*
  - Example:



B. Juliano © 2002.

43

### Copyright and Intellectual Property Notice

This document and its contents are the *Intellectual Property* (IP) of Dr. Benjoe A. Juliano of the Department of Computer Science at California State University, Chico (CSUC). Dr. Juliano claims exclusive moral rights of ownership under current *Copyright Laws* (Title 17 of the United States Code and 1998 Digital Millennium Copyright Act) and *IP Policies/Guidelines* (CSUC EM83-08, EM97-07, and Article 39 of the CFA/CSU Contract) including, but not limited to:

- the exclusive right to copy, reproduce, and/or distribute this document;
- the right to be identified as the creator of this work (the right of attribution);
- the right to take action against false attribution; and
- the right to object to derogatory treatment of this work (the right of integrity).

B. Juliano © 2002, based on notes by J. Ullman



43