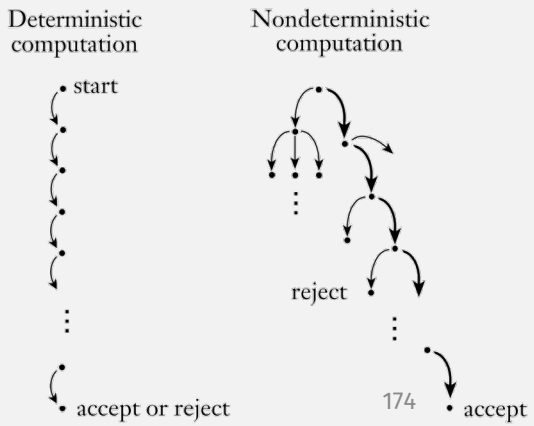
Nondeterminism



Logistics

- HW1 Coming due.
- Questions?

A Brief Intro to XML

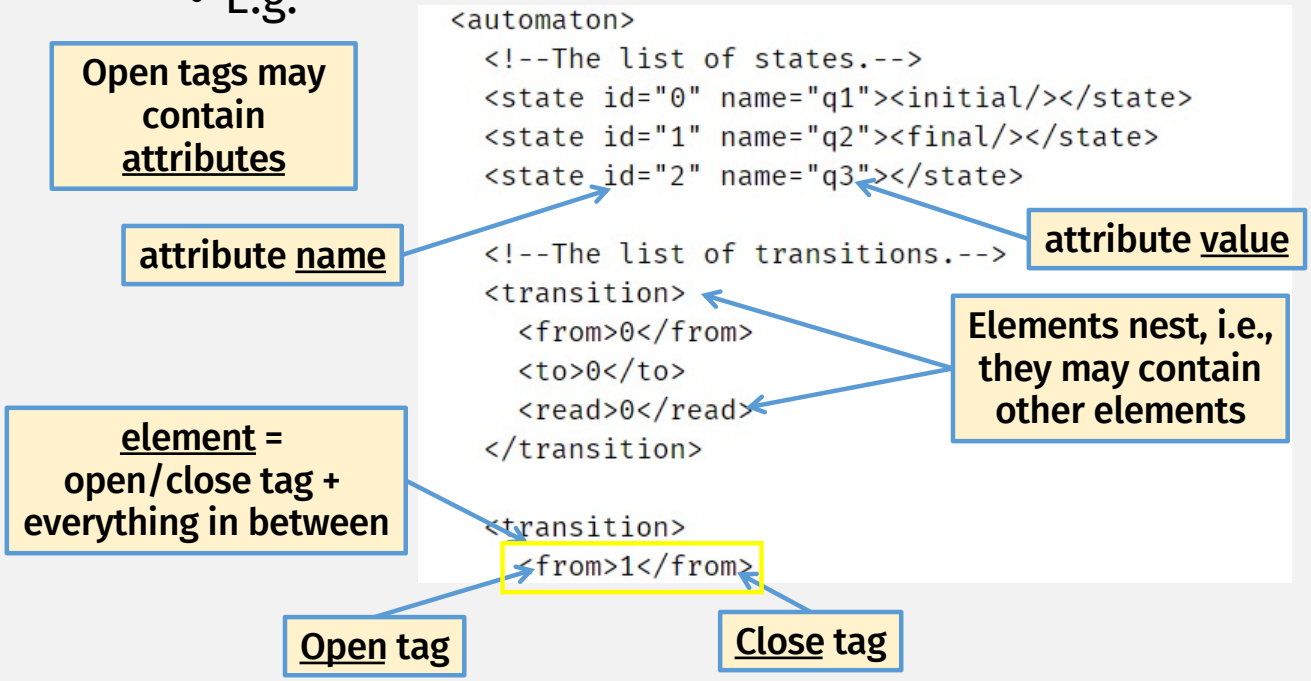
- What is it?
  - It's a widely-used "data interchange format"
    - I.e., A standard, language-agnostic way for programs to send/recv data
  - (JSON is another popular interchange format)

- Example, when querying web apis:
  - `https://api.etrade.com/v1/market/quote/GOOG`

```
<?xml version="1.0" encoding="UTF-8"?>
<QuoteResponse>
   <QuoteData>
      <dateTime>15:17:00 EDT 06-20-2018</dateTime>
      <dateTimeUTC>1529522220</dateTimeUTC>
      <quoteStatus>DELAYED</quoteStatus>
      <ahFlag>false</ahFlag>
      <hasMiniOptions>false</hasMiniOptions>
      <All>
         <adjustedFlag>false</adjustedFlag>
         <annualDividend>0.0</annualDividend>
         <ask>1175.79</ask>
         <askExchange />
         <askSize>100</askSize>
         <askTime>15:17:00 EDT 06-20-2018</askTime>
         <bid>1175.29</bid>
         <bidExchange />
         <bidSize>100</bidSize>
```

XML in this class: 2 purposes

1. Grader uses it to send/get state machines to/from HW
   - E.g.




```
<automaton>
  <!--The list of states.-->
  <state id="0" name="q1"><initial/></state>
  <state id="1" name="q2"><final/></state>
  <state id="2" name="q3"></state>

  <!--The list of transitions.-->
  <transition>
    <from>0</from>
    <to>0</to>
    <read>0</read>
  </transition>

  <transition>
    <from>1</from>
```

XML in this class: 2 purposes

A ***language*** is a set of strings.

M ***recognizes language*** A
if $A = \{w | M \text{ accepts } w\}$

2. Running example of a "language", to compare/contrast computation models
   - E.g.,

```
<automaton>
  <!--The list of states.-->
  <state id="0" name="q1"><initial/></state>
  <state id="1" name="q2"><final/></state>
  <state id="2" name="q3"></state>

  <!--The list of transitions.-->
  <transition>
    <from>0</from>
    <to>0</to>
    <read>0</read>
  </transition>

  <transition>
    <from>1</from>
```

"Language" of all possible open tag strings is regular

"Language" of all XML strings is <u>not</u> regular, because a DFA cannot do open/close tag matching

We're already familiar with a non-regular language! Stay tuned!

Last time: "Closed" Operations

A set is **closed** under an operation if applying the operation to members of the set returns an element still in the set

- E.g., Natural numbers = {0, 1, 2, …}
  - closed under addition,
  - not closed under subtraction

Last time: Union is Closed for Reg. Langs

THEOREM 1.25

The class of regular languages is closed under the union operation.

In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Proof (implement this algorithm for HW2)

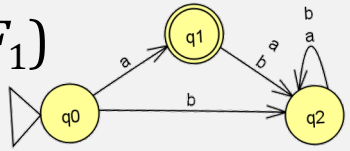
- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize A_1,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize A_2,
- Construct a new machine $M = (Q, \Sigma, \delta, q_0, F)$ using M_1 and M_2

M simulates running its input on both M_1 and M_2 in "parallel"; accept if either accepts

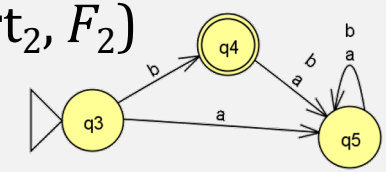
DFA Union Example

- A_1 = {“a”}, A_2 = {“b”}, $A_1 \cup A_2$ = {“a”, ”b”}

- $M_1 = (Q_1, \Sigma, \delta_1, \text{start}_1, F_1)$



$M_2 = (Q_2, \Sigma, \delta_2, \text{start}_2, F_2)$



- M recognizing {“a”, “b”} = $(Q, \Sigma, \delta, \text{start}, F)$
  - $Q = Q_1 \times Q_2$ = {(q0, q3), (q0, q4), (q0, q5), …}
  - Σ = {a, b}
  - $\delta((q0, q3), a) = (\delta_1(q0), \delta_2(q3)) = (q1, q5)$
    - …
  - start = (q0, q3)
  - F = {(q1, q3), (q1, q4), (q1, q5), (q0, q4), …}

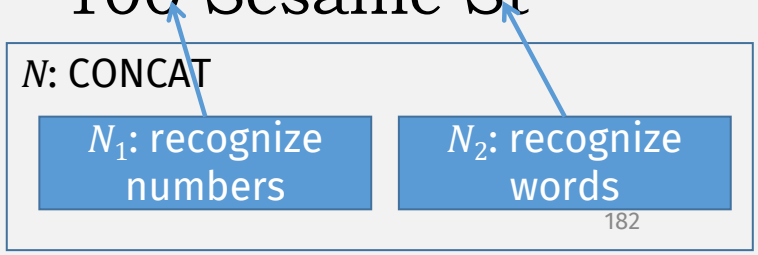
Last time: Is Concatenation Closed?

THEOREM 1.26

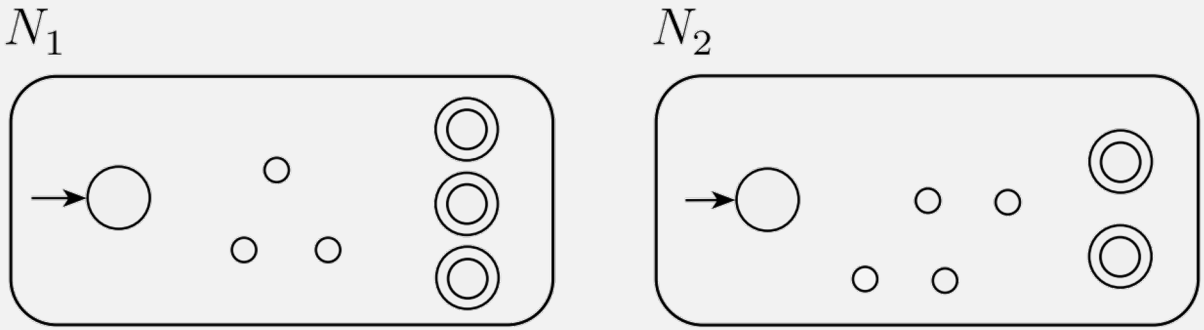
The class of regular languages is closed under the concatenation operation.

In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

- Proof: Construct a new machine? (like union)
- How does N know when to switch from N_1 to N_2?
  - Can only read input once



Concatentation



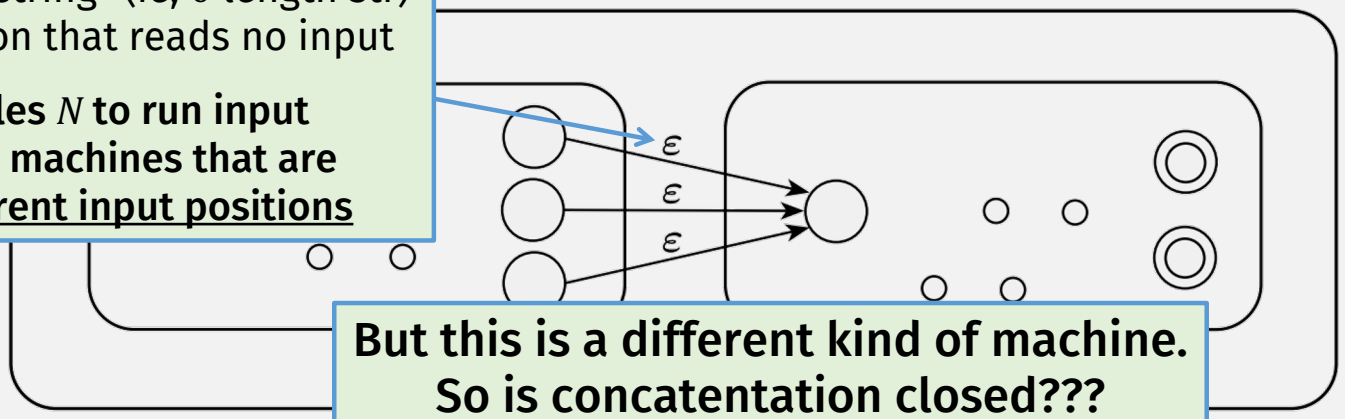
Let N_1 recognize A_1, and N_2 recognize A_2.

Want: Construction of N to recognize $A_1 \circ A_2$

N must simultaneously:
- Keep checking with N_1 **and**
- Move to N_2 to check 2nd part

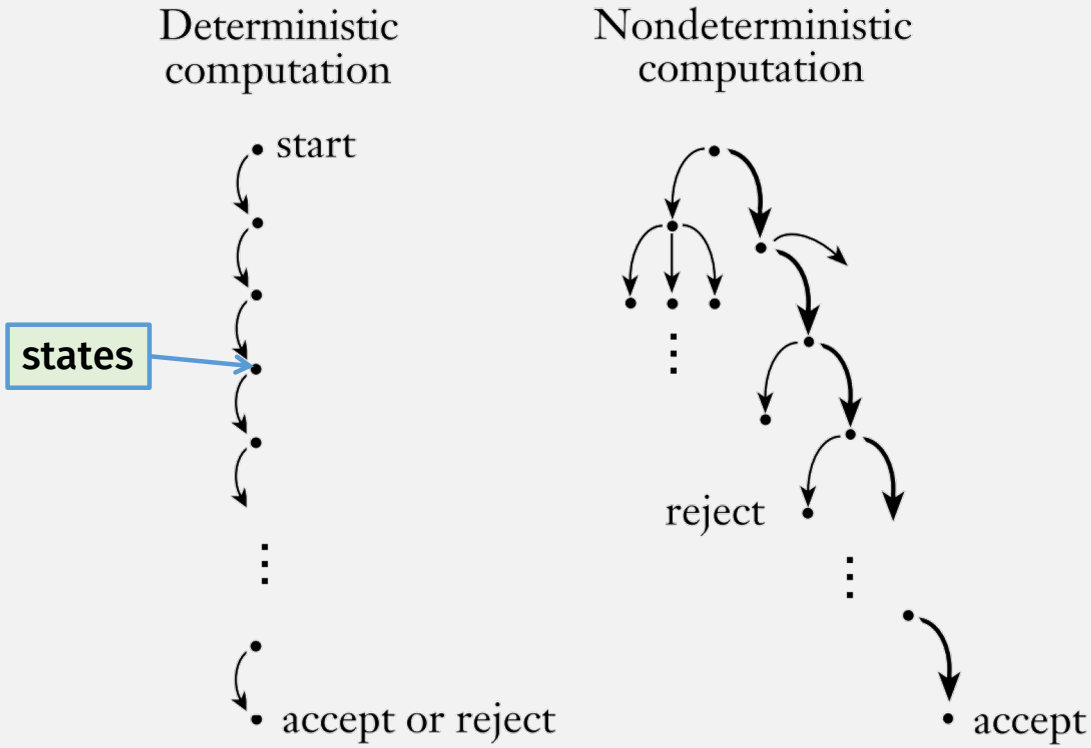
ε = "empty string" (ie, 0 length str)
= transition that reads no input

Enables N to run input on two machines that are at different input positions



But this is a different kind of machine. So is concatentation closed???

Nondeterminism



Nondeterministic machine can be in multiple states at once

DEFINITION 1.37

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Power set

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Power Sets

- A power set is the set of all subsets of a set
- Example: `S = {a,b,c}`
- Power set of S =
  - `{{},{a},{b},{c},{a,b},{a,c},{b,c},{a,b,c}}`

Formal Definition of "Computation"

- DFA (from before): Let $w = w_1 w_2 \cdots w_n$

M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

- NFA: Let $w = y_1 y_2 \cdots y_m$

N ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_m$ exists in Q with three conditions:

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \ldots, m-1$, and
3. $r_m \in F$.

This is now a set

Nondeterministic computation requires **only one path to accept state** in the computation tree

Note:

- δ(q,s) may map to ∅ some *q* and *s* (*what does that mean*?)
- δ(q,s) may map to a set containing multiple *q*'s
- A string is said to be accepted *if there exists* a path from q_0 to some state in *F*
- A string is rejected *if there exist NO path* to any state in *F*

- Equivalently,
  L(M) = {w | w is in Σ^* and w is accepted by M}

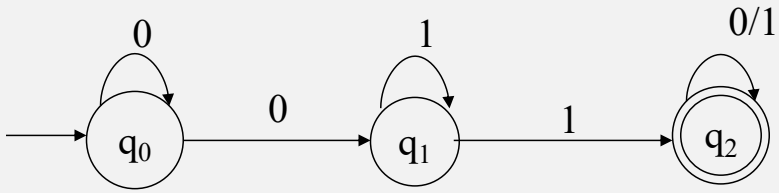
- Example #1: one or more 0's followed by one or more 1's

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

Start state is q_0

$F = \{q_2\}$

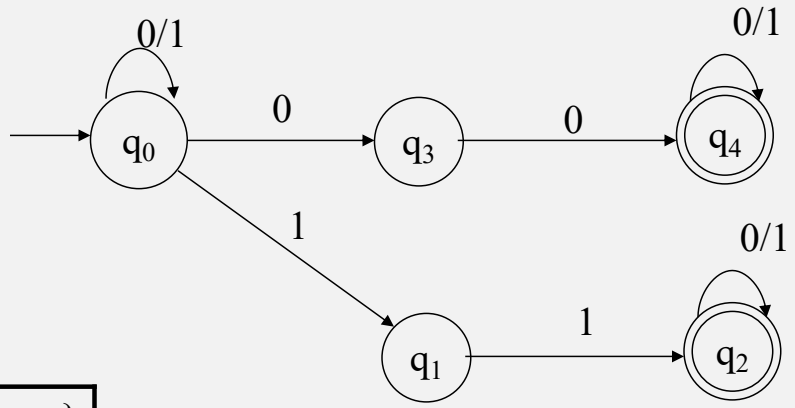


δ:

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

- Example #2: pair of 0's *or* pair of 1's as substring

$Q = \{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma = \{0, 1\}$
Start state is q_0
$F = \{q_2, q_4\}$



δ:

	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	$\{\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	$\{\}$
q_4	$\{q_4\}$	$\{q_4\}$

Epsilon transitions

Nondeterministic machine can be in multiple states at once

DEFINITION 1.37

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

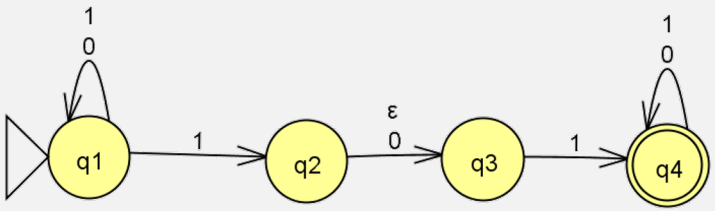
1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is th…te, and
5. $F \subseteq Q$ is the set of accept states.

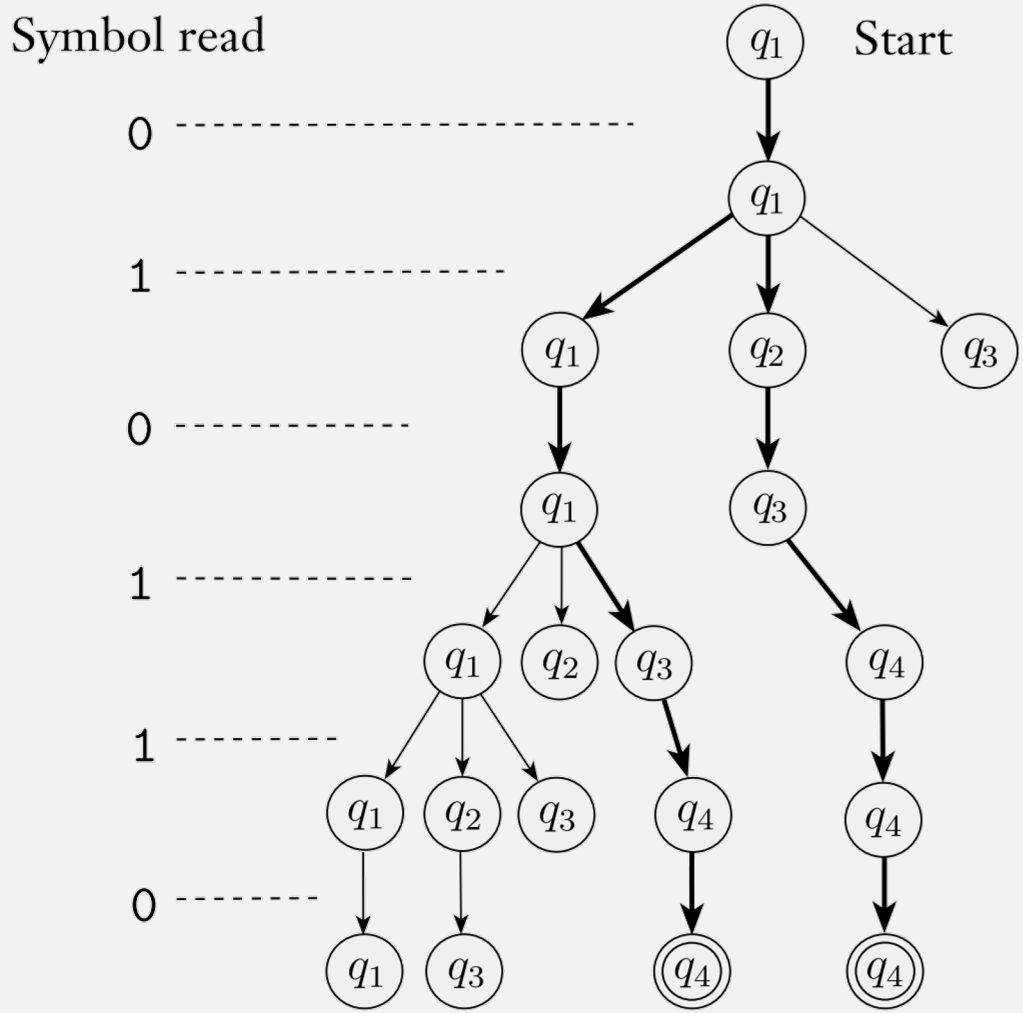
Includes ε

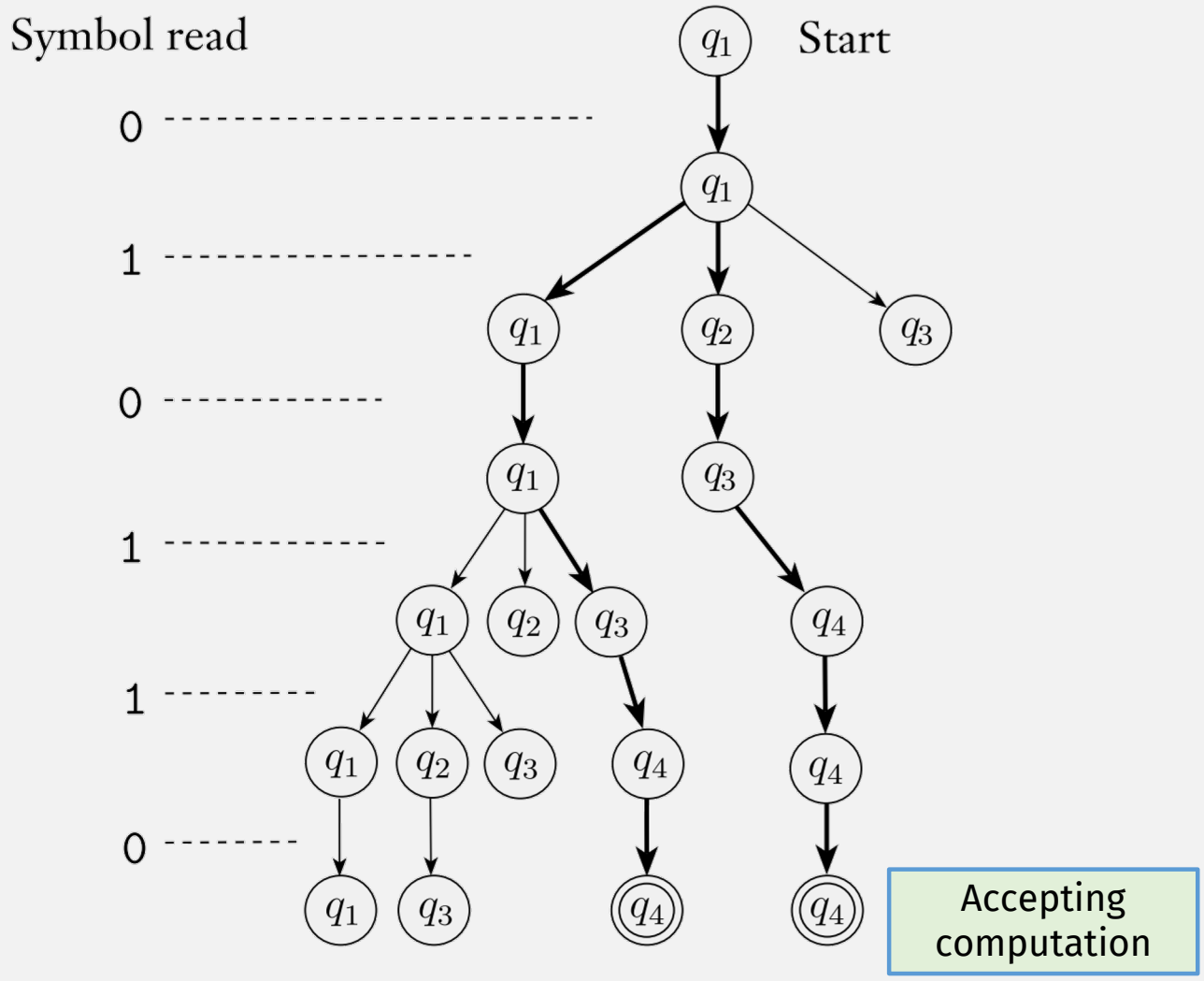
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Example Fig 1.27 (JFLAP demo): **010110**

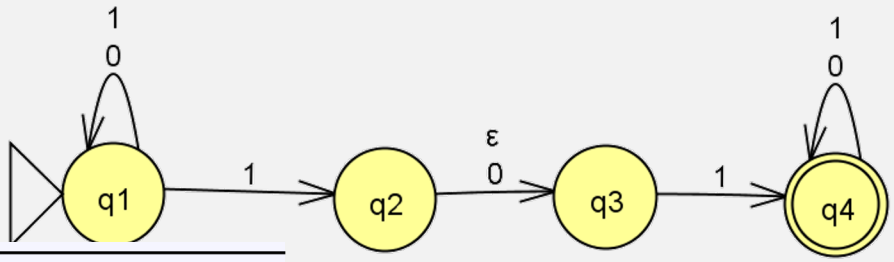



Symbol read
Start
0
1
0
1
1
0
q_1
q_2
q_3
q_4


Symbol read
Start
0
1
0
1
1
0
Accepting computation

In-class exercise

- Come up with a formal description of the following NFA:



DEFINITION 1.37

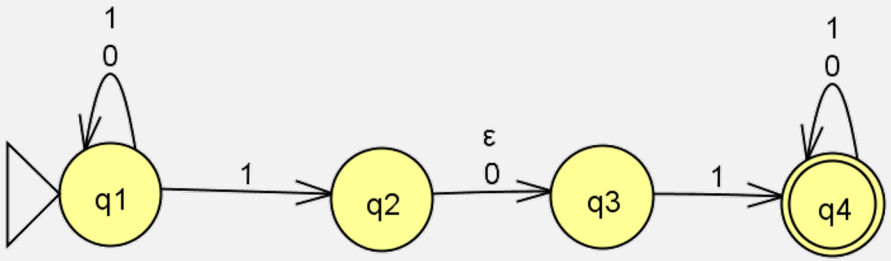
A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

4. q_1 is the start state, and
5. $F = \{q_4\}$.



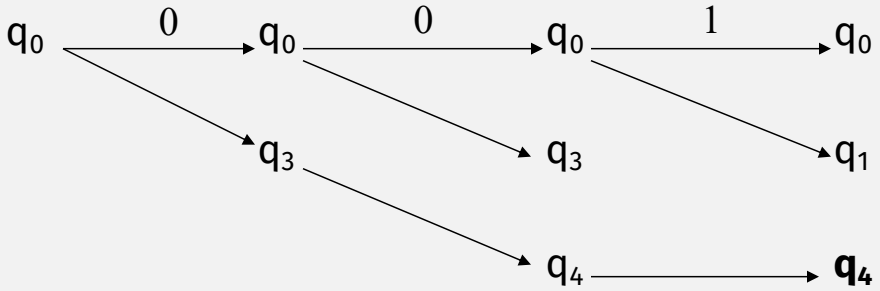
Thoughts and Questions

- How does an NFA find the correct/accepting path for a given string?
  - NFAs are a non-intuitive computing model
  - You could use *backtracking* to find if there exists a path to a final state (following slide)
- Why NFA?
  - We are *primarily* interested in NFAs as language defining capability, i.e., do NFAs accept languages that DFAs do not?
  - Other secondary questions include practical ones such as whether or not NFA is easier to develop, or how does one implement NFA

Question: Why non-determinism is useful?

- Non-determinism = Backtracking
- Compressed information
- Non-determinism hides backtracking
- Programming languages, e.g., Prolog, hides backtracking => Easy to program at a higher level: *what we want to do, rather than how to do it*
- Useful in algorithm complexity study

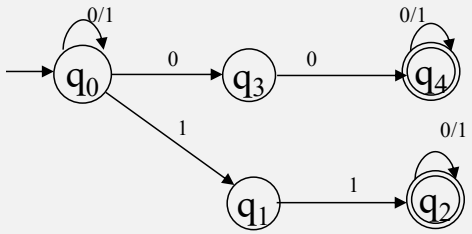
- Thus, determining if a given NFA (example #2) accepts a given string (001) can be done algorithmically:



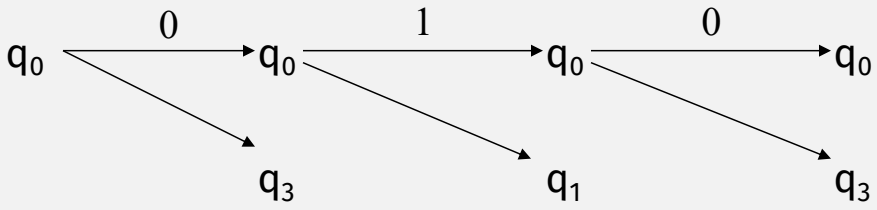
accepted

- Each level will have at most *n* states:

  Complexity: O(|x|*n), for running over a string *x*

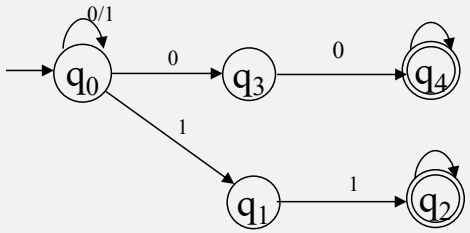


- Another example (010):



not accepted

- All paths have been explored, and none lead to an accepting state.



Sketch out NFA “run” fn pseudocode

(Aside)
Extension of δ to Strings and Sets of States

What we currently have: $\delta : (Q \times \Sigma_\varepsilon) \rightarrow 2^Q$

What we might want (why?): $\delta : (2^Q \times \Sigma^*) \rightarrow 2^Q$

So. Is Concatenation Closed for Reg Langs?

- Concatenation of DFAs produces an NFA
- But:

> A language is called a ***regular language*** if some DFA recognizes it.

- To show that concatenation is closed for regular languages, we must prove that NFAs *also* recognize regular languages.
- Specifically, we must prove:
  - NFAs ⇔ regular languages

Proving NFAs recognize the regular langs

- *Theorem*:
  - A language A is regular **if and only if** some NFA N recognizes it.

How to Prove $X \Leftrightarrow Y$ ---*"Ping Pong"*

- $X \Leftrightarrow Y$ = "X if and only if Y" = X iff Y
- Proof <u>at minimum</u> has 2 parts:

1. **=>** if X, then Y
   - i.e., assume X, then use it to prove Y
   - "forward" direction
2. **<=** if Y, then X
   - i.e., assume Y, then use it to prove X
   - "reverse" direction

Proving NFAs recognize the regular langs

- *Theorem*:
  - A language A is regular **if and only if** some NFA N recognizes it.

- Must prove:
  - **=>** If A is regular, then some NFA N recognizes it

  - **<=** If an NFA *N* recognizes A, then A is regular.

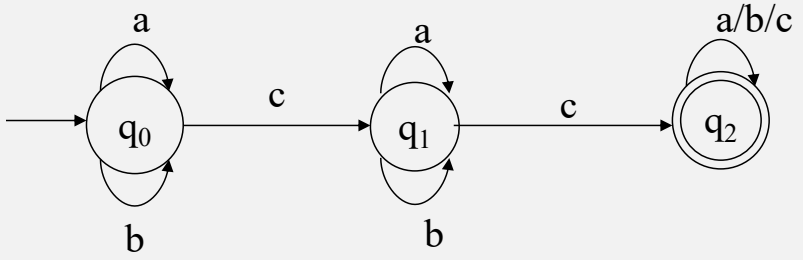
In other words, equivalence of DFAs & NFAs?

- *Do* DFAs and NFAs accept the same *class* of languages?
  - Is there a language L that is accepted by a DFA, but not by any NFA?
  - Is there a language L that is accepted by an NFA, but not by any DFA?

- Observation: Every DFA is an NFA, DFA is only restricted NFA.

- Consider the following DFA: 2 or more c's

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b, c\}$
Start state is q_0
$F = \{q_2\}$



δ:

	a	b	c
q_0	q_0	q_0	q_1
q_1	q_1	q_1	q_2
q_2	q_2	q_2	q_2

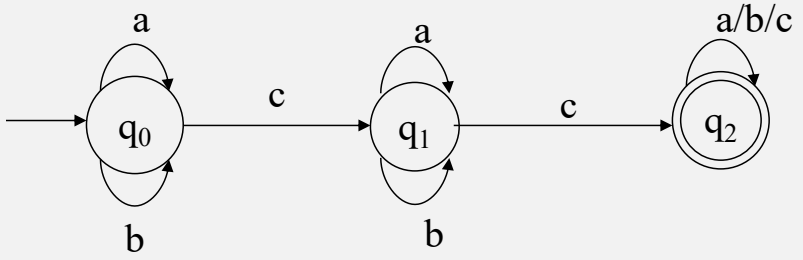
- An Equivalent NFA:

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b, c\}$

Start state is q_0

$F = \{q_2\}$



δ:

	a	b	c
q_0	$\{q_0\}$	$\{q_0\}$	$\{q_1\}$
q_1	$\{q_1\}$	$\{q_1\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$

- **Lemma 1:** Let M be an DFA. Then there exists a NFA M' such that L(M) = L(M').

- **Proof:** Every DFA is an NFA. Hence, if we let M' = M, then it follows that L(M') = L(M).

  The above is just a formal statement of the observation from the previous slide.

- Therefore, if L is a regular language then there exists an NFA M such that L = L(M).

- It follows that NFAs accept all regular languages.

- But do NFAs accept more?

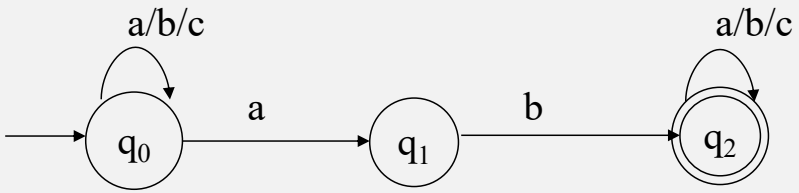
Which is to ask:

Is NDA more “powerful” than DFA?

i.e., accepts type of languages that any DFA cannot?

Example Demonstrating the question

- Let Σ = {a, b, c}. Give an NFA M that accepts: L = {*x* | *x* is in Σ^* and *x* contains *ab*}

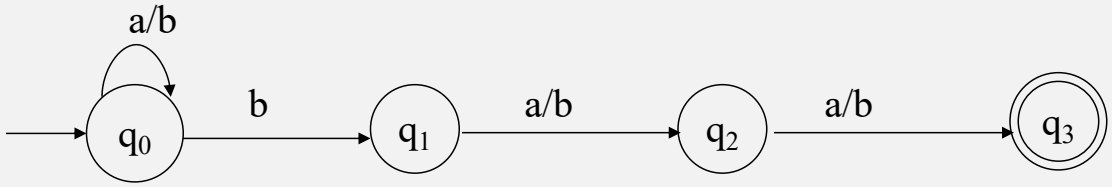


1. Is L a subset of L(M)? Or, does M accepts all string in L?
2. Is L(M) a subset of L? Or, does M rejects all strings not in L?

- **Is an NFA necessary? Can we draw a DFA for this L?**
- Designing NFAs is not as trivial as it seems: easy to create bug accepting string outside language

- Let Σ = {a, b}. Give an NFA M that accepts:

  L = {*x* | *x* is in Σ^* and the third to the last symbol in *x* is *b*}



  Is L a subset of L(M)?

  Is L(M) a subset of L?

- **You all: Give an equivalent DFA**

Proving NFAs recognize the regular langs

- *Theorem*:
  - A language A is regular **if and only if** some NFA N recognizes it.

- Must prove:
  - **=>** If A is regular, then some NFA N recognizes it ✓
    - Easy
    - We know: if A is regular, then a **DFA** recognizes it.
    - Easy to convert DFA to an NFA! (how?)
  - **<=** If an NFA *N* recognizes A, then A is regular.
    - Hard
    - Idea: Convert NFA to DFA
    - **Next time!**

Check-in Quiz

On Gradescope