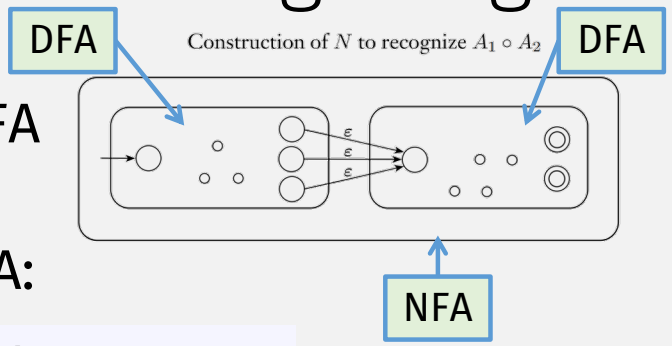
NFA -> DFA
and
Intro to Regular Expressions

Logistics

- HW2: due Oct 1
- **Questions**?

Last time: Is Concat Closed for Reg Langs?



- Concatentation of DFAs produces an NFA
- But, regular lang defined using only DFA:

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

- To show: *Concatenation is closed for regular languages*, we must prove that NFAs *also* recognize regular languages.
- Specifically:
  - Theorem (1.40): NFAs ⇔ regular languages

Last time: How to Prove a Theorem: $X \Leftrightarrow Y$

- $X \Leftrightarrow Y$ = "X if and only if Y" = X iff Y = X <=> Y
- Proof at minimum 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

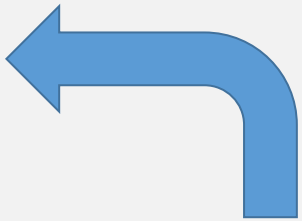
- *Theorem*:
  - A language A is regular **if and only if** some NFA N recognizes it.

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

How to convert NFA -> DFA?

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***.



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.

Proof idea:
Each "state" of the DFA must be a set of states in the NFA

Example:

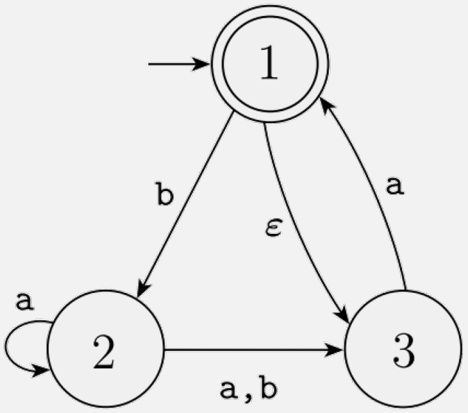


FIGURE **1.42**
The NFA N_4

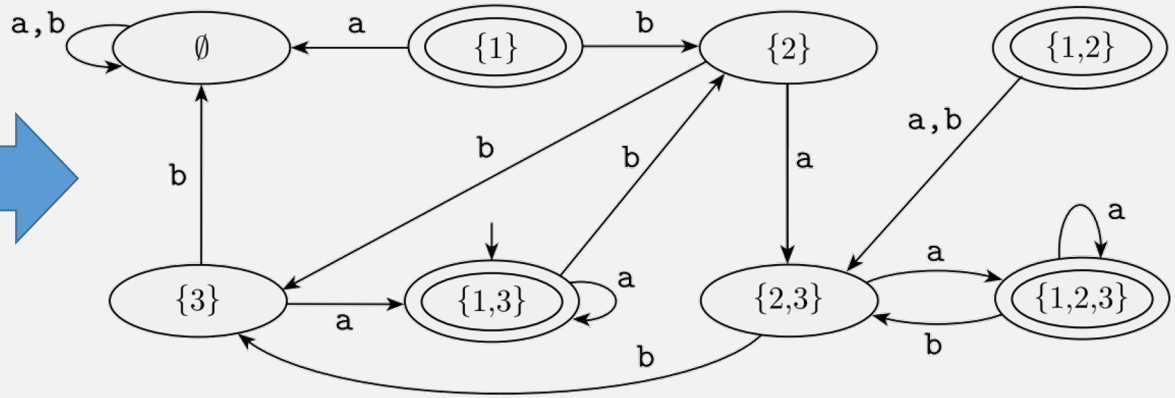
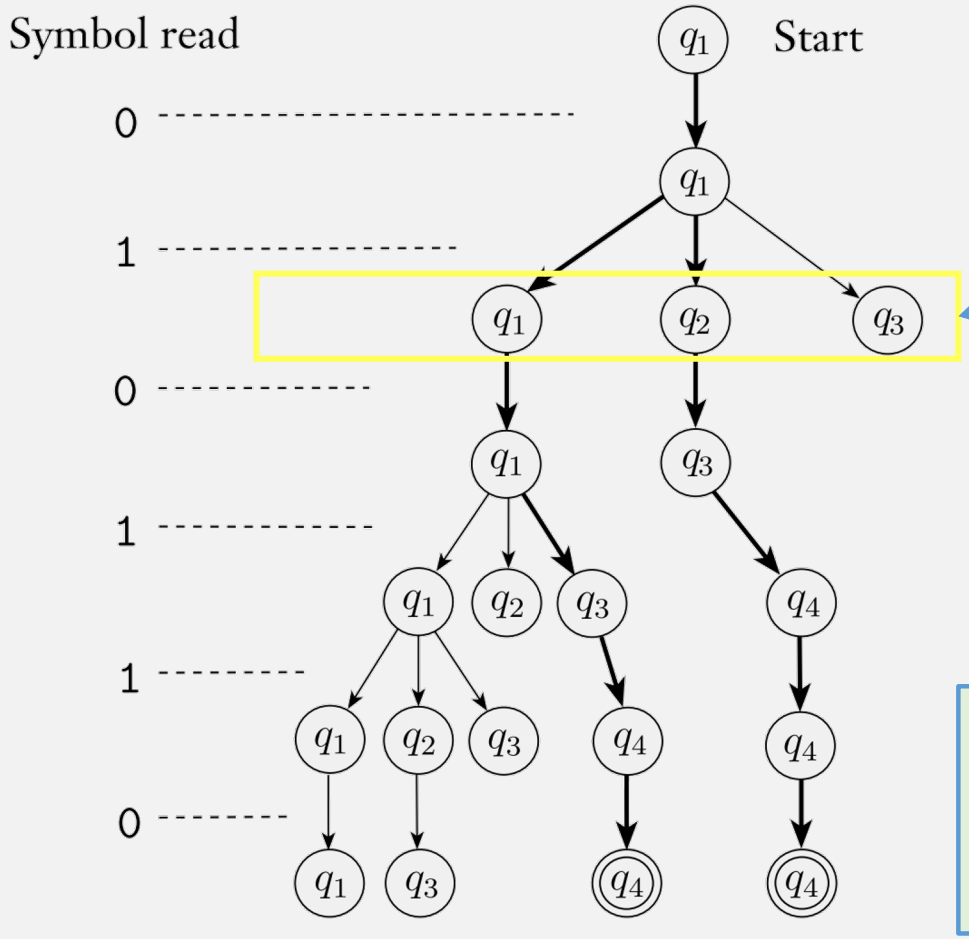


FIGURE **1.43**
A DFA D that is equivalent to the NFA N_4



In a DFA, all these states at each step must be only **one** state

So design a state in the DFA to be a **set of NFA states**!

This is a generalization of the proof strategy from Thm 1.25 (closure of union), where a state = pair of "states"

Converting NFA -> DFA, Formally

- **A key outstanding lemma**: Let N be an NFA. Then there exists a DFA M' such that L(N) = L(M').
- Let NFA N = $(Q, \Sigma, \delta, q_0, F)$
- An equivalent DFA M' has states Q' = $\mathcal{P}(Q)$ (power set of Q)

Two stages:

First consider NFAs without epsilon transitions

Then, we'll consider NFAs with epsilon transitions

Convert NFA w/o ε -> DFA, Formally

Lemma: Let N be an NFA w/o ε transitions. Then there exists a DFA M' such that L(N) = L(M').

NFA -> DFA (ignore empty transitions)

- Have: $N = (Q, \Sigma, \delta, q_0, F)$
- Want to: construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$

1. $Q' = \mathcal{P}(Q)$.

A state for M is a set of states in N

2. For $R \in Q'$ and $a \in \Sigma$,

R = a state in M = a set of states in N

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

To compute next state for R,
compute next states of each NFA state r in R,
then union results into one set

3. $q_0' = \{q_0\}$

4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

- Example: L(N) = empty string, or start and end with 0

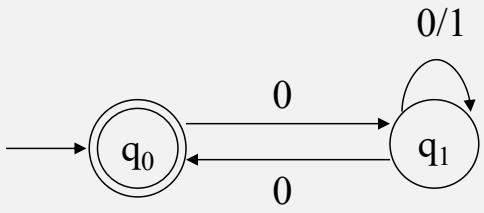
$Q = \{q_0, q_1\}$

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

$\delta = \ldots$

$q_0 = q_0$

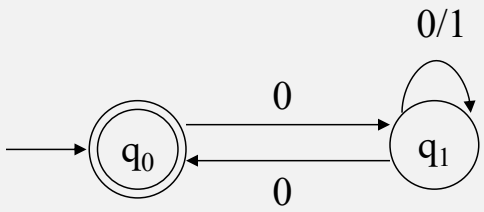
$F = \{q_0\}$



δ:

	0	1
-->q_0	$\{q_1\}$	$\{\}$
q_1	$\{q_0, q_1\}$	$\{q_1\}$

- Example of creating a DFA out of an NFA w/o ε (as per the constructive proof):



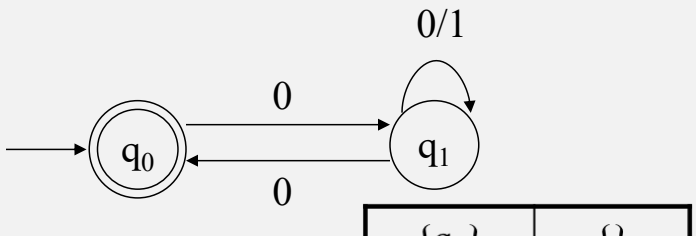
-->q_0	$\{q_1\}$	$\{\}$
q_1	$\{q_0, q_1\}$	$\{q_1\}$

δ for DFA:

	0	1
->[q_0]	$\{q_1\}$ write as [q_1]	$\{\}$ write as [ ]
[q_1]		
[ ]		

Can get confusing set vs state:
Answer: [] or () to distinguish!

- Example of creating a DFA out of an NFA w/o ε (as per the constructive proof):

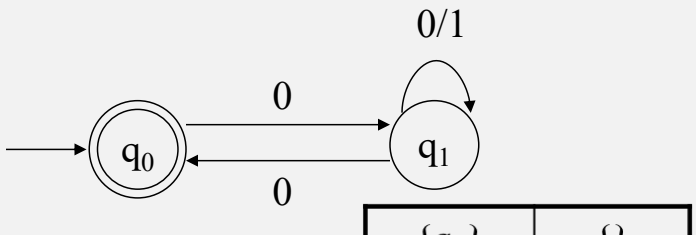


$\{q_1\}$	$\{\}$
$\{q_0, q_1\}$	$\{q_1\}$

δ:

	0	1
->$[q_0]$	$\{q_1\}$ write as $[q_1]$	$\{\}$
$[q_1]$	$\{q_0,q_1\}$ write as $[q_{01}]$	$\{q_1\}$
[ ]		
$[q_{01}]$		

- Example of creating a DFA out of an NFA w/o ε (as per the constructive proof):

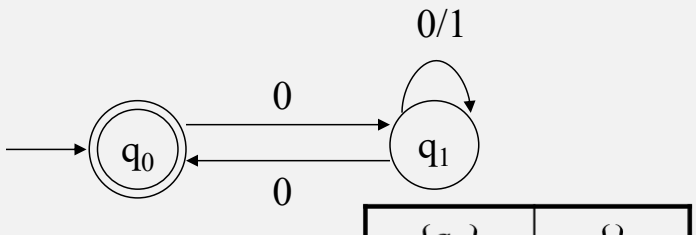


$\{q_1\}$	$\{\}$
$\{q_0, q_1\}$	$\{q_1\}$

δ:

	0	1
->q_0	$\{q_1\}$ write as $[q_1]$	$\{\}$
$[q_1]$	$\{q_0,q_1\}$ write as $[q_{01}]$	$\{q_1\}$
[ ]	[ ]	[ ]
$[q_{01}]$		

- Example of creating a DFA out of an NFA w/o ε (as per the constructive proof):

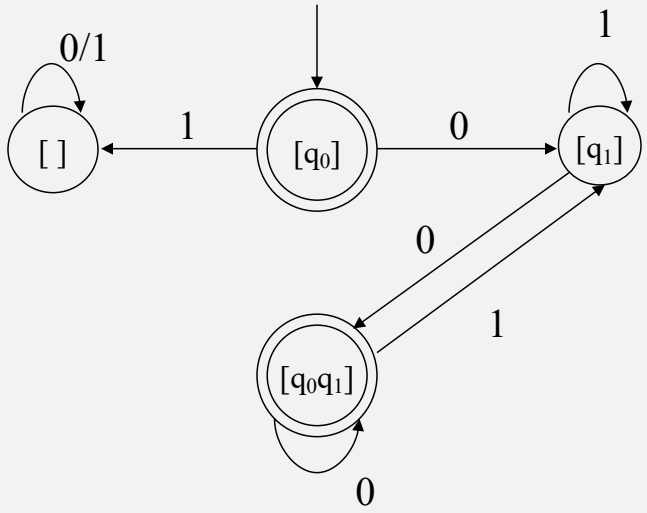


$\{q_1\}$	$\{\}$
$\{q_0, q_1\}$	$\{q_1\}$

δ:

	0	1
->q_0	$\{q_1\}$ write as $[q_1]$	$\{\}$
$[q_1]$	$\{q_0,q_1\}$ write as $[q_{01}]$	$\{q_1\}$
[ ]	[ ]	[ ]
$[q_{01}]$	$[q_{01}]$	$[q_1]$

- Construct DFA M' as follows:



$\delta(\{q_0\}, 0) = \{q_1\}$	=>	$\delta'([q_0], 0) = [q_1]$
$\delta(\{q_0\}, 1) = \{\}$	=>	$\delta'([q_0], 1) = [\ ]$
$\delta(\{q_1\}, 0) = \{q_0, q_1\}$	=>	$\delta'([q_1], 0) = [q_0q_1]$
$\delta(\{q_1\}, 1) = \{q_1\}$	=>	$\delta'([q_1], 1) = [q_1]$
$\delta(\{q_0, q_1\}, 0) = \{q_0, q_1\}$	=>	$\delta'([q_0q_1], 0) = [q_0q_1]$
$\delta(\{q_0, q_1\}, 1) = \{q_1\}$	=>	$\delta'([q_0q_1], 1) = [q_1]$
$\delta(\{\}, 0) = \{\}$	=>	$\delta'([\ ], 0) = [\ ]$
$\delta(\{\}, 1) = \{\}$	=>	$\delta'([\ ], 1) = [\ ]$

L(DFA) ⬅➔L(NFA w/o ε)

- **Theorem:** Let L be a language. Then there exists an DFA M such that L = L(M) iff there exists an NFA (w/o ε) N' such that L = L(N').

- **Proof:**

  (if) Suppose there exists an NFA (w/o ε) N' such that L = L(N'). Then by the prior Lemma there exists an DFA M such that L = L(M).

  (only if) Suppose there exists an DFA M such that L = L(M). Then by Lemma from last time there exists an NFA (w/o ε) N' such that L = L(N').

QED!

- **Corollary:** The NFAs, w/o ε, define the regular languages.

Remember

- Suppose R = {}. Then empty transition out

$$\delta(R, 0) = \bigcup_{q \in R} \delta(q, 0)$$
$$= \{\}$$

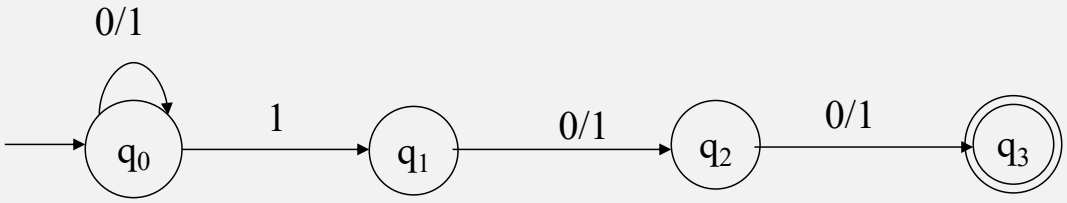
Since R = {}

In-class Exercise: An NFA State Diagram

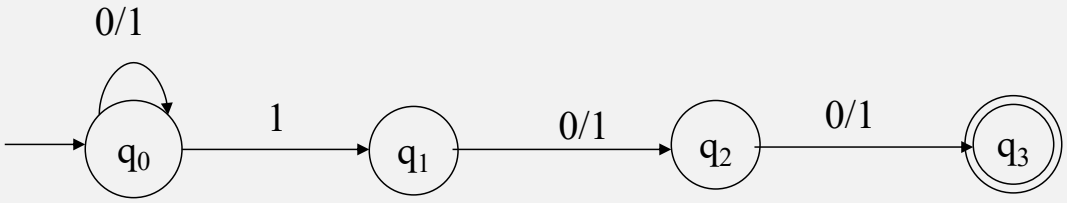
$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{0, 1\}$
Start state is q_0
$F = \{q_0\}$

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

- In-class Exercise 2 : Describe the language of this machine



- In-class Exercise 2 : Describe the language of this machine



1. Could you convert this NFA to a DFA?
2. How annying/easy would that be?

NFAs with ε Moves

- An NFA-ε is a five-tuple: $M = (Q, \Sigma, \delta, q_0, F)$

  δ A transition function, which is a total function from $Q \times \Sigma \cup \{\varepsilon\}$ to 2^Q

  $\delta: (Q \times (\mathbf{\Sigma \cup \{\varepsilon\}})) \rightarrow 2^Q$

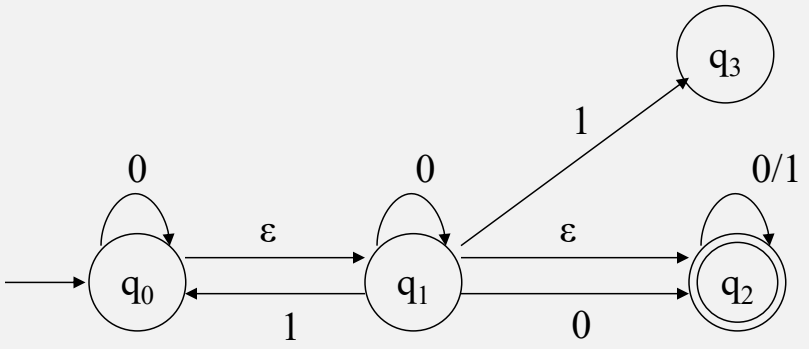
  δ(q,s) -The set of all states p such that there is a transition labeled a from q to p, where a is in $\Sigma \cup \{\varepsilon\}$

- Sometimes referred to as an NFA-ε other times, simply as an NFA.

Informal Definitions

- Let N = (Q, Σ, δ,q_0,F) be an NFA-ε.
- A String w in Σ^* is *accepted* by M iff there exists a path in N from q_0 to a state in F labeled by w **and zero or more ε transitions.**
- The language accepted by N is the set of all strings from Σ^* that are accepted by N.

- Example:



δ:	0	1	ε
q_0	$\{q_0\}$	$\{\ \}$	$\{q_1\}$
q_1	$\{q_1, q_2\}$	$\{q_0, q_3\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$	$\{\ \}$
q_3	$\{\ \}$	$\{\ \}$	$\{\ \}$

- A string $w = w_1w_2...w_n$ is processed as $w = \varepsilon^*w_1\varepsilon^*w_2\varepsilon^* ... \varepsilon^*w_n\varepsilon^*$
- Example: all computations on 00:

0 ε 0

q_0 q_0 q_1 q_2

:

ε-closure, aka E(·)

- Define ε-closure(q) to denote the set of all states reachable from q by zero or more ε transitions.

- Examples:

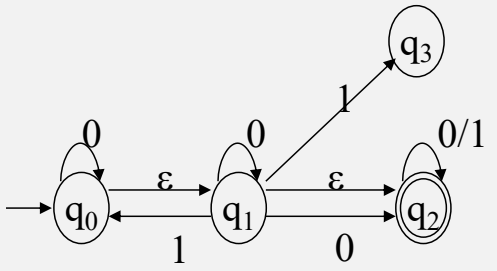
  $E(q_0) = \{q_0, q_1, q_2\}$ $\quad$ $E(q_2) = \{q_2\}$
  $E(q_1) = \{q_1, q_2\}$ $\quad$ $E(q_3) = \{q_3\}$

- E(q) can be extended to sets of states by defining:

$$E(P) = \bigcup_{q \in P} E(q)$$

- Examples:

  $E(\{q_1, q_2\}) = \{q_1, q_2\}$
  $E(\{q_0, q_3\}) = \{q_0, q_1, q_2, q_3\}$



NFA -> DFA (with empty transitions)

$E(R) = \{q|\ q$ can be reached from R by traveling along 0 or more ε arrows$\}$

- Have: $N = (Q, \Sigma, \delta, q_0, F)$
- Want to: construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$

1. $Q' = \mathcal{P}(Q)$.

2. For $R \in Q'$ and $a \in \Sigma$,

$$\delta'(R, a) = \bigcup_{r \in R} ~~\delta(r, a)~~ E(\delta(r, a))$$

For each r, do its transition in N, then add states reachable from empty transitions, then union results into one set

3. $q_0' = ~~\{q_0\}~~ E(\{q_0\})$

4. $F' = \{R \in Q' |\ R$ contains an accept state of $N\}$

Definitions for NFA-ε Machines

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA-ε and let w be in Σ^*. Then w is *accepted* by N iff $\delta\hat{}=E(\delta(\{q_0\}, w))$ contains at least one state in F.

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA-ε. Then the *language accepted* by N is the set:

  $L(N) = \{w \mid w \text{ is in } \Sigma^* \text{ and } \delta\hat{}(\{q_0\}, w) \text{ contains at least one state in F}\}$

- Another equivalent definition:

  $L(N) = \{w \mid w \text{ is in } \Sigma^* \text{ and w is accepted by N}\}$

Relationships b/t classes of machine

- DFA =?= NFA w/ε =?= NFA w/o ε
- L of a DFA <=> L of an NFA

Equivalence of NFAs and NFA-εs?

- Do NFAs and NFA-ε machines accept the same *class* of languages?

    1. Is there a language L that is accepted by a NFA, but not by any NFA-ε?

    2. Is there a language L that is accepted by an NFA-ε, but not by any DFA?

Observation: Every NFA is an NFA-ε.

- **Lemma 1:** Let N be an NFA. Then there exists a NFA-ε N' such that L(N) = L(N').
- **Proof:** Every NFA is an NFA-ε. Hence, if we let N' = N, then it follows that L(N') = L(N).

- Therefore, if L is a regular language then there exists an NFA-ε N such that L = L(N).
- It follows that NFA-ε machines accept all regular languages.
- **So far: Regular <-> L(DFA) <-> L(NFA) -> L(NFA-ε).**
- But do NFA-ε machines accept more?

The “Pong” part

- **Lemma 2:** Let N be an NFA-ε. Then there exists a NFA N' such that L(N) = L(N').

- **Proof:** (sketch)

  Let M = (Q, Σ, δ,q_0,F) be an NFA-ε.

  Define an NFA N' = (Q, Σ, δ',q_0,F') as:

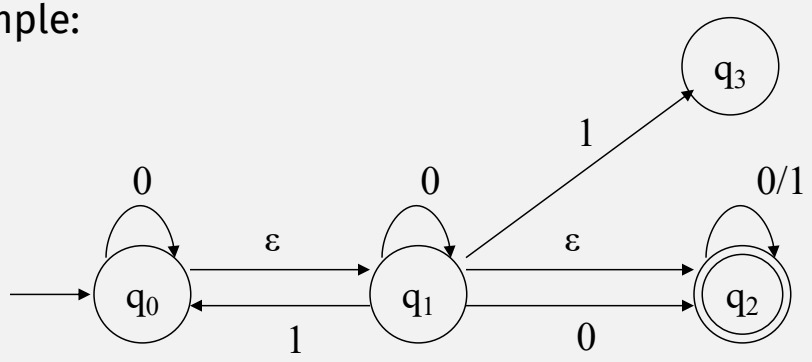
  F' = F ∪ {q} if E(q) contains at least one state from F

  F' = F otherwise

  δ'(q, a) = δ^(q, a) - for all q in Q and a in Σ

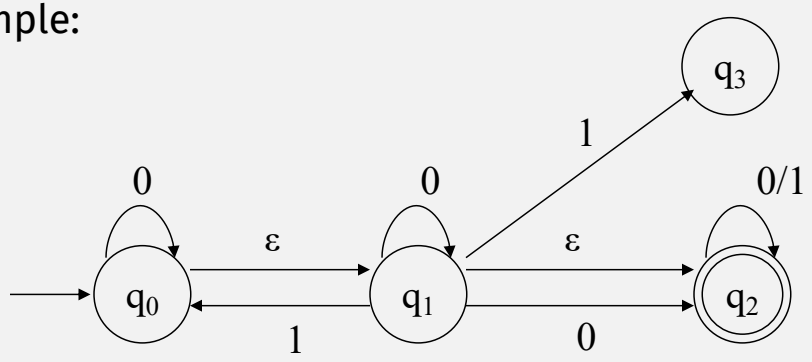
- Notes:
  - δ': (Q x Σ) –> 2^Q is a function
  - N' has the same state set, the same alphabet, and the same start state as N
  - N' has no ε transitions

- Example:



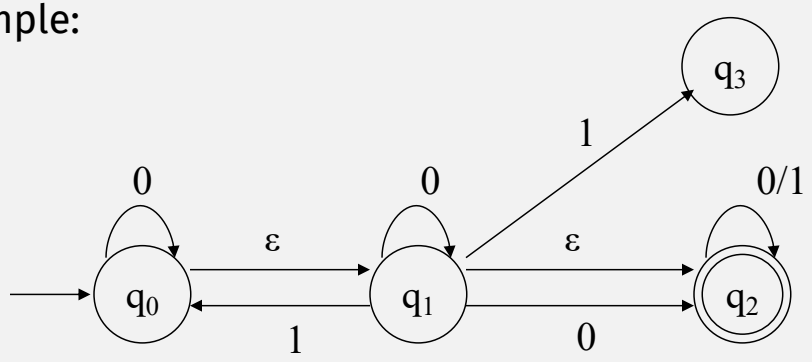
- Step #1:
  - Same state set as N
  - q_0 is the starting state

- Example:

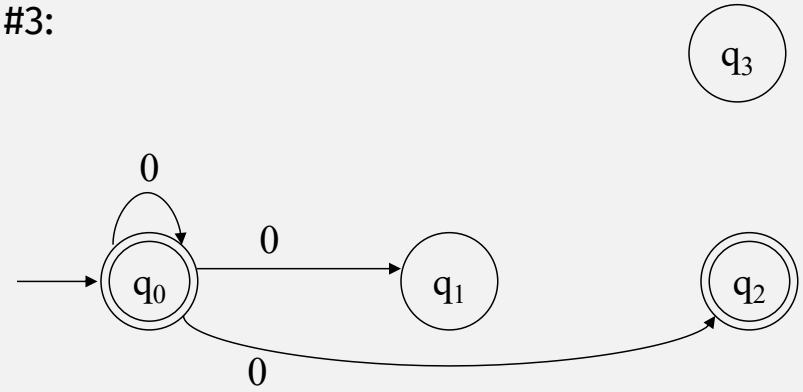


- Step #2:
  - q_0 becomes a final state

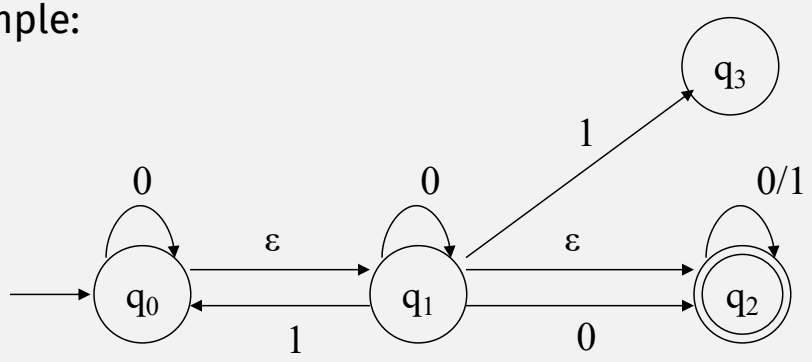
- Example:



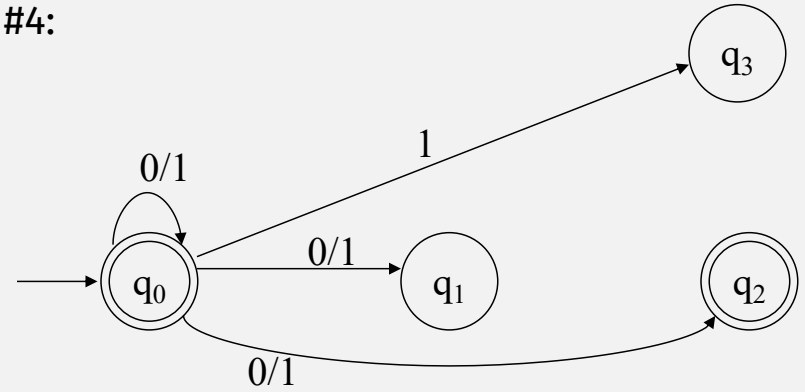
- Step #3:



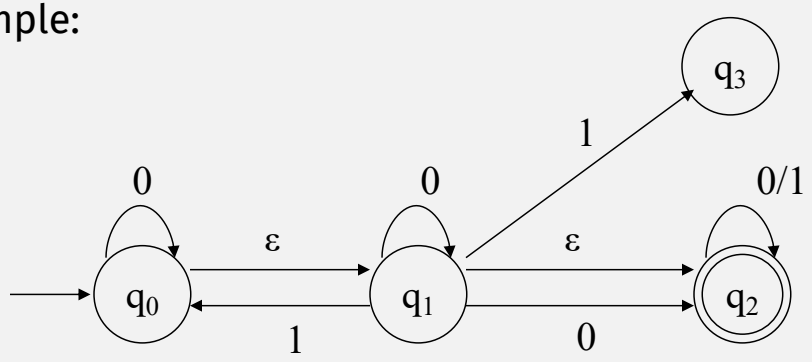
- Example:



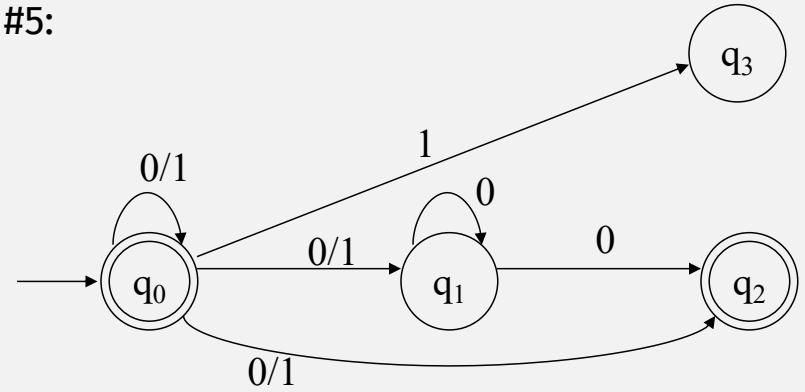
- Step #4:



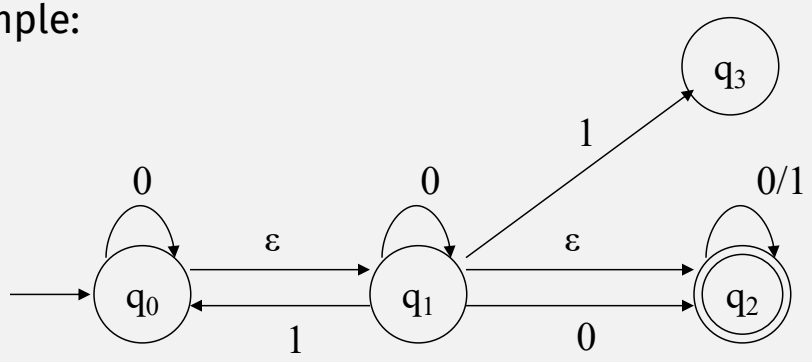
- Example:



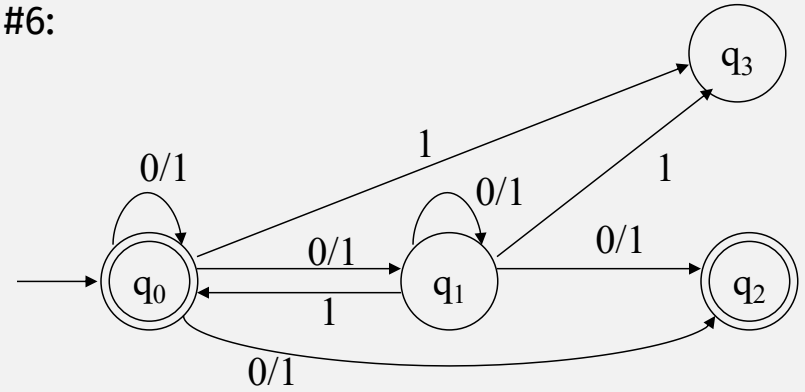
- Step #5:



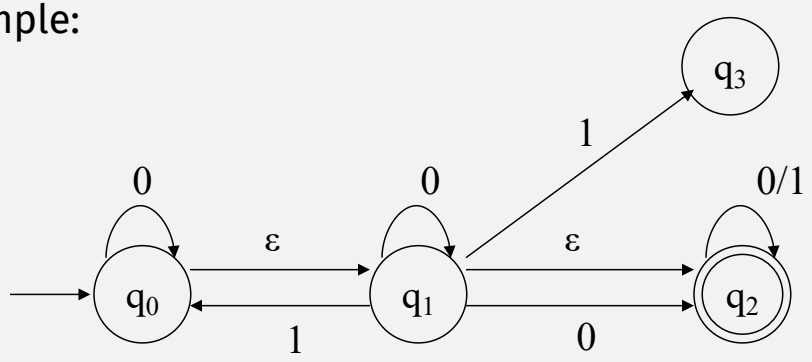
- Example:



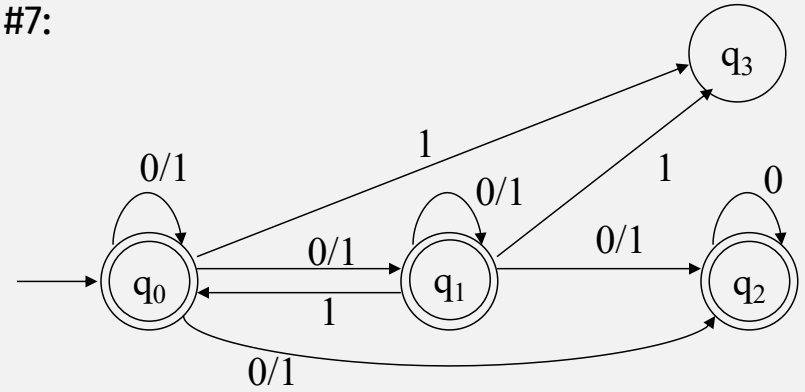
- Step #6:



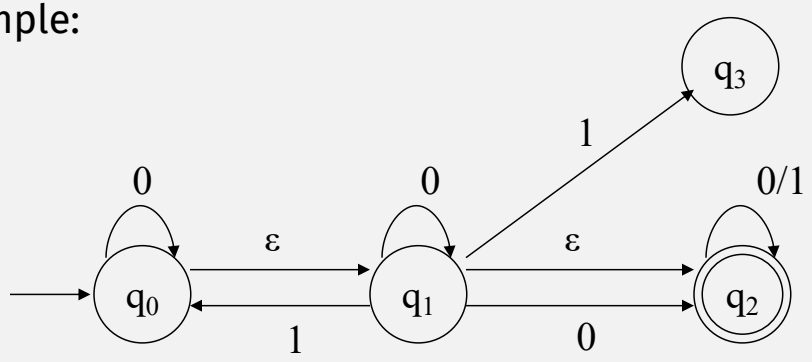
- Example:



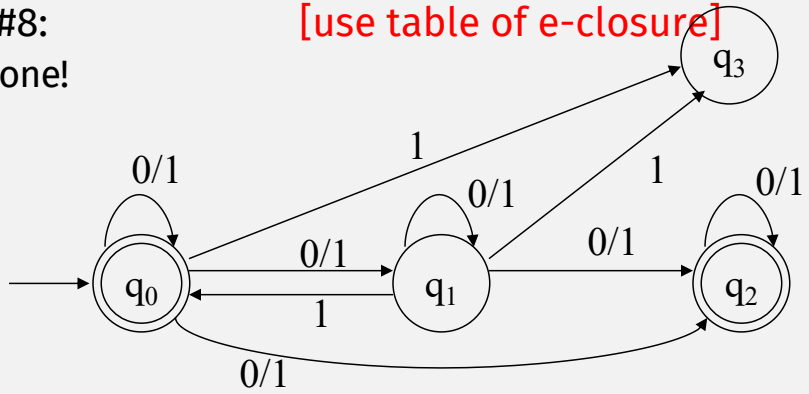
- Step #7:



- Example:



- Step #8: [use table of e-closure]
  - Done!



- **Theorem:** Let L be a language. Then there exists an NFA N such that L= L(N) iff there exists an NFA-ε N' such that L = L(N').

- **Proof:**

  (if) Suppose there exists an NFA-ε N' such that L = L(N'). Then by Lemma 2 there exists an NFA N such that L = L(N).

  (only if) Suppose there exists an NFA N such that L = L(N). Then by Lemma 1 there exists an NFA-ε N' such that L = L(N').

- **Corollary:** The NFA-ε machines define the regular languages.

Proving that NFAs Recognize Reg Langs

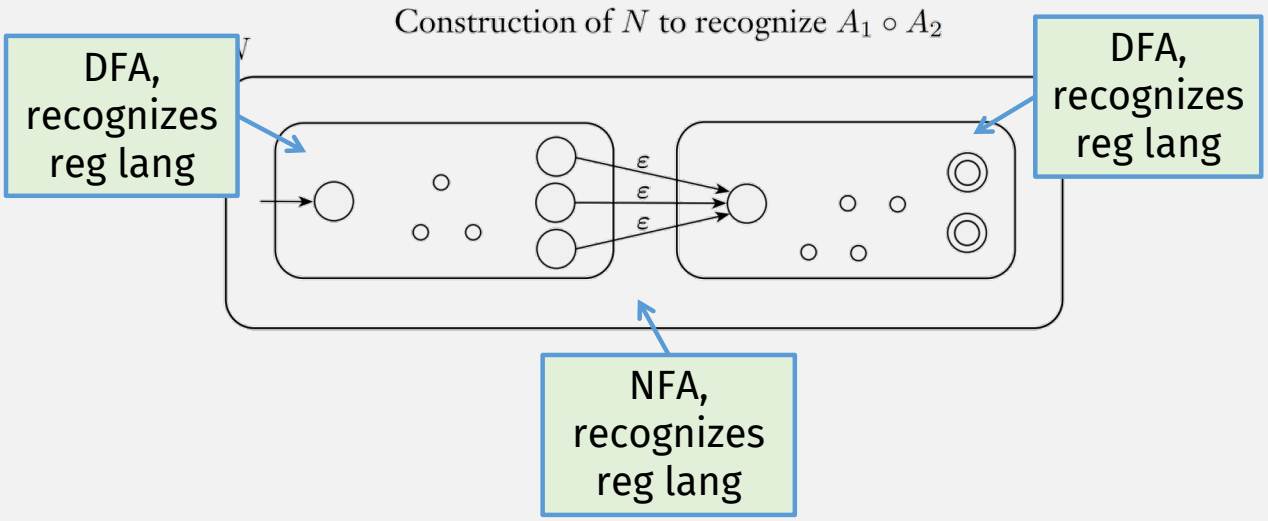
- *Theorem*:
  - A language A is regular **if and only if** some NFA N recognizes it.

- Must prove:
  - **=>** If A is regular, then some NFA N recognizes it
    - Easy
    - We know: if A is regular, then a DFA recognizes it
    - Convert DFA to an NFA
  - **<=** If an NFA *N* recognizes A, then A is regular
    - Hard
    - We know: if a DFA recognizes a lang, then it is regular
    - Idea: Convert NFA to DFA
    - Using NFA -> DFA algorithm we just created!

■ (Q.E.D.)

So Concatenation is Closed for Reg Langs!

- Concatentation of DFAs produces an NFA

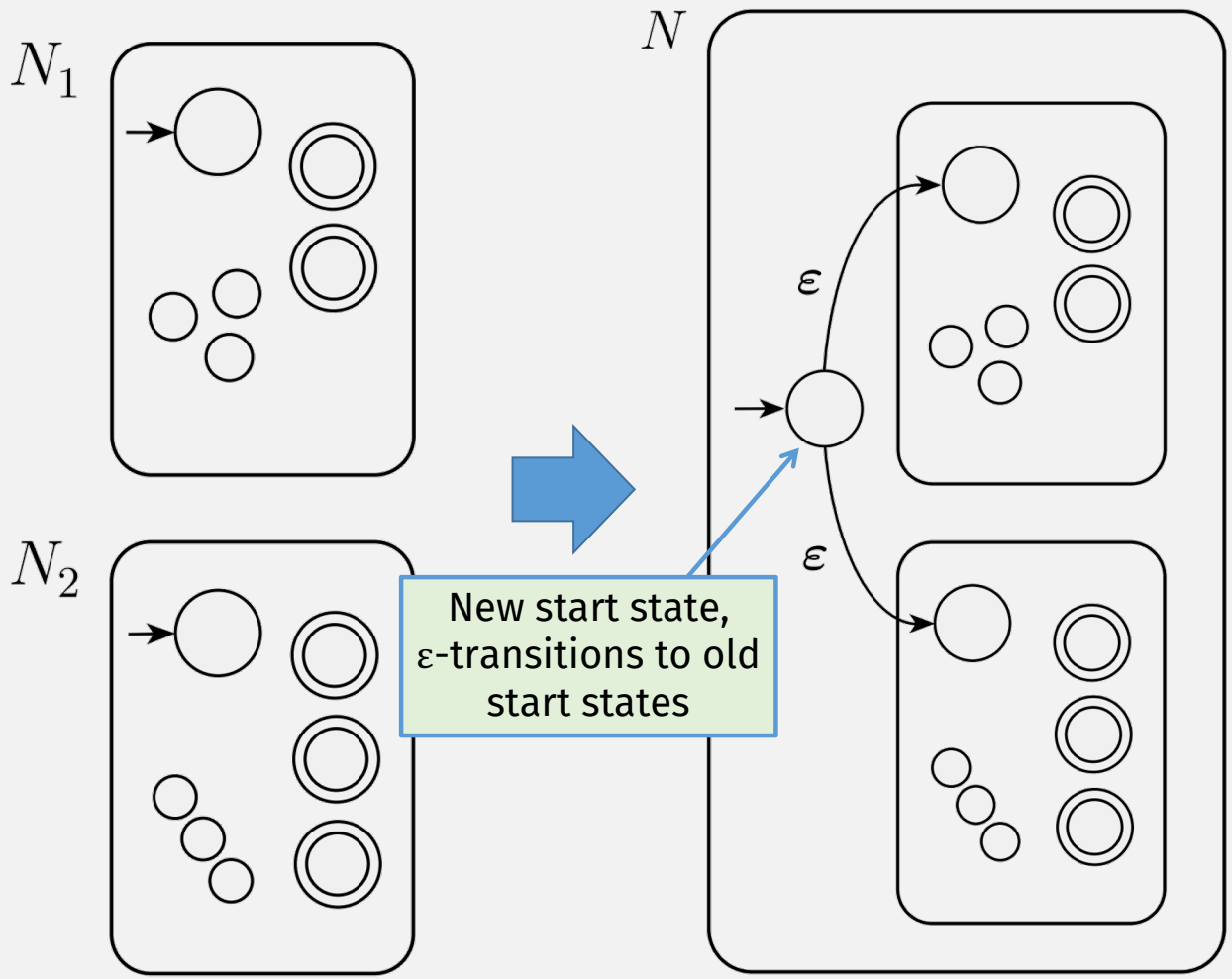


“Regular” Operations

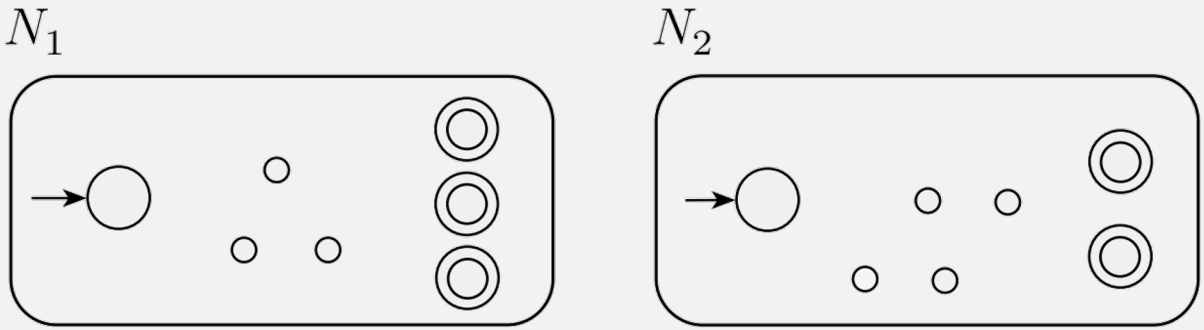
- Regular languages are closed under these operations:
  - Union (already proved with DFAs)
  - Concatenation
  - Kleene Star (repetition, zero or more times)

- Easier to prove closure (by construction) using NFAs

Union

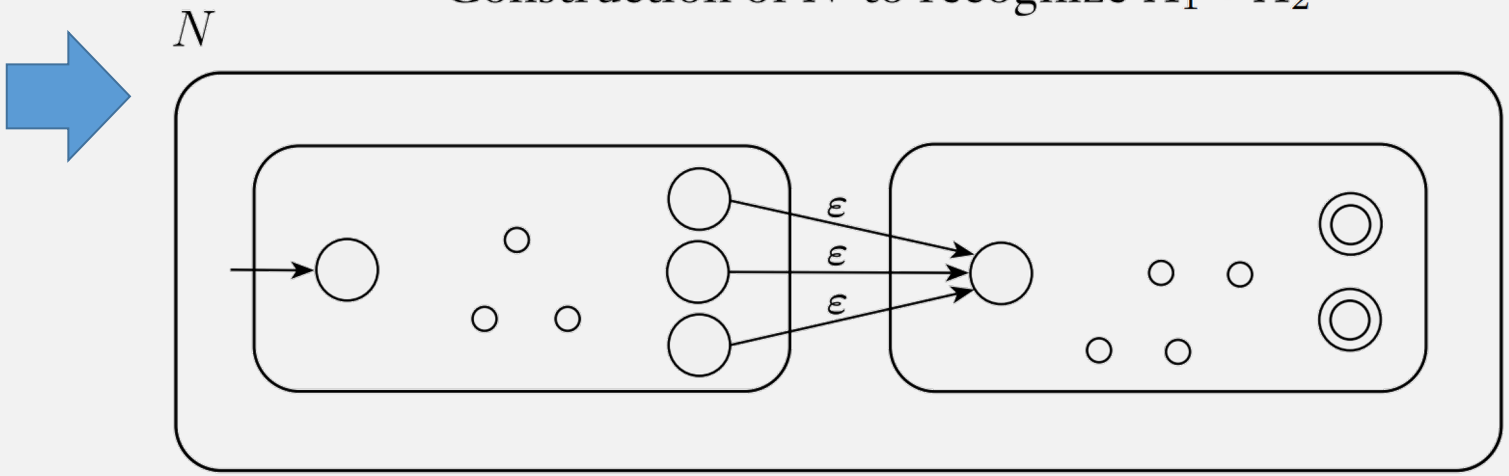

N_1
N_2
N
ε
ε
New start state, ε-transitions to old start states

Concat

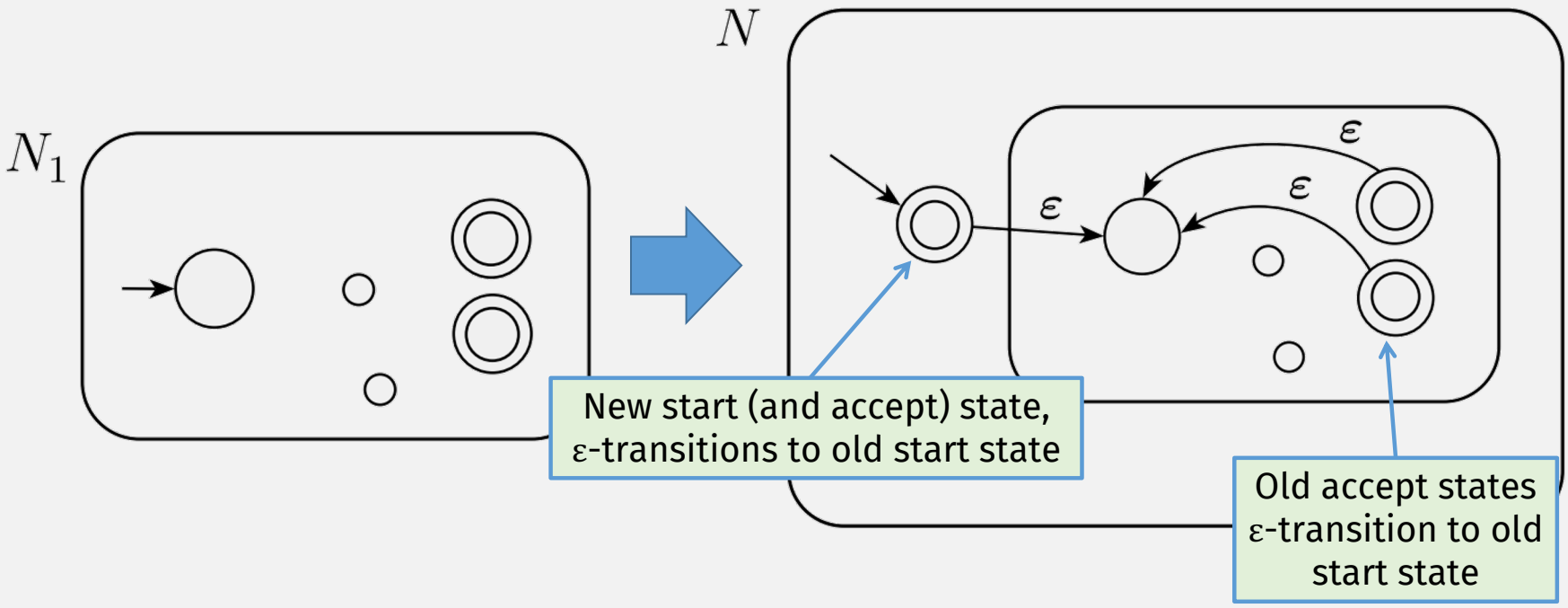


Let N_1 recognize A_1, and N_2 recognize A_2.

Construction of N to recognize $A_1 \circ A_2$



Kleene Star


N_1
N
ε
ε
ε
New start (and accept) state, ε-transitions to old start state
Old accept states ε-transition to old start state

Why do we care?

- Union, concat, and Kleene star represent all regular languages
- I.e., they define **regular expressions**

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

union → 4.
concat → 5.
star → 6.

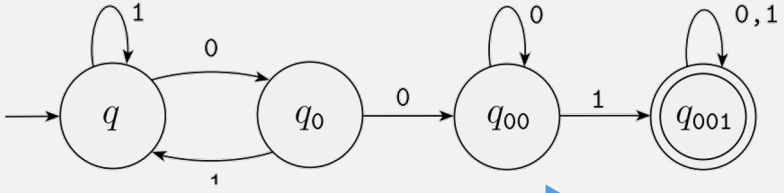
Regexes

- Originally developed for modelling biological systems
- Adopted for use in string matching
- Popularized in Unix system tools
  - 1986 C Regex Library
  - 1987 Perl
  - 1988 GNU Regex library

- Following Perl, adopted into most modern languages:

Ways to Recognize a Regular Language

- Instead of:



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

- Or:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

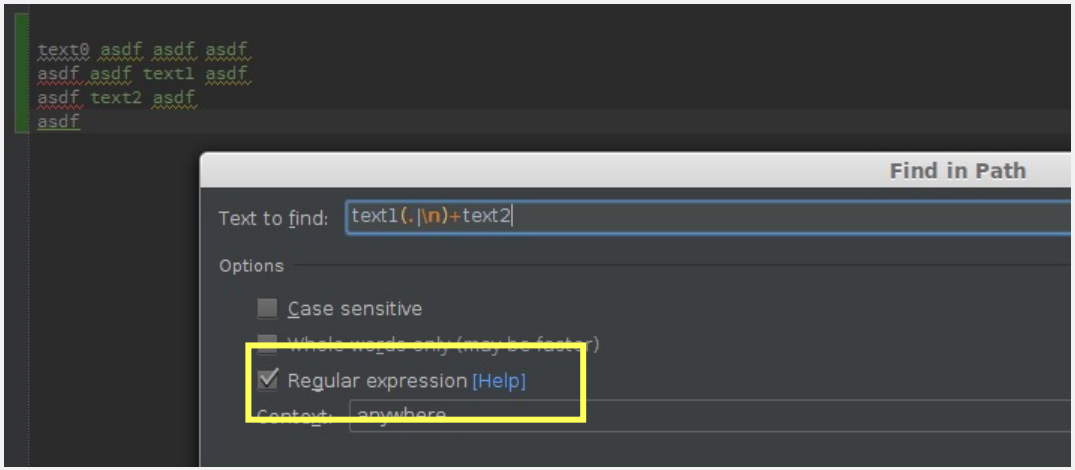
- We can write a regexp: $\Sigma^*001\Sigma^*$

These all define a computer (program) that accepts all strings containing `001`

Which would you rather write?

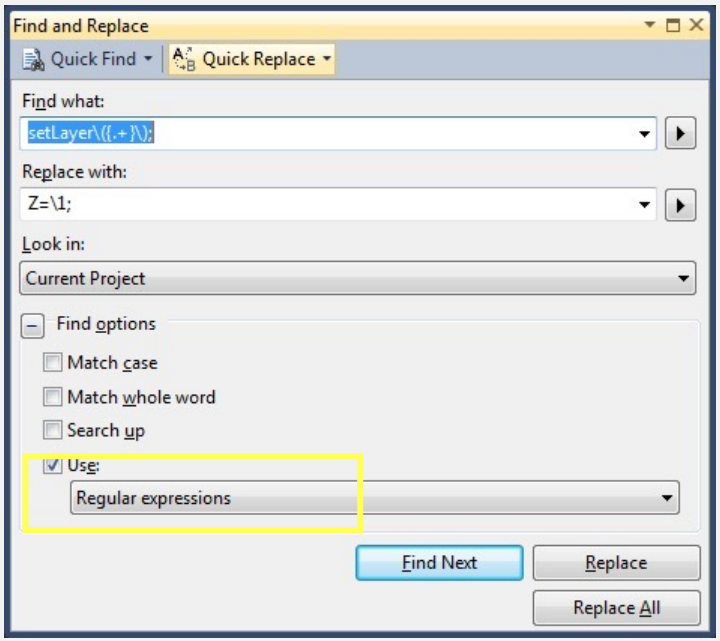
Regular Expressions are Super Useful

- IntelliJ



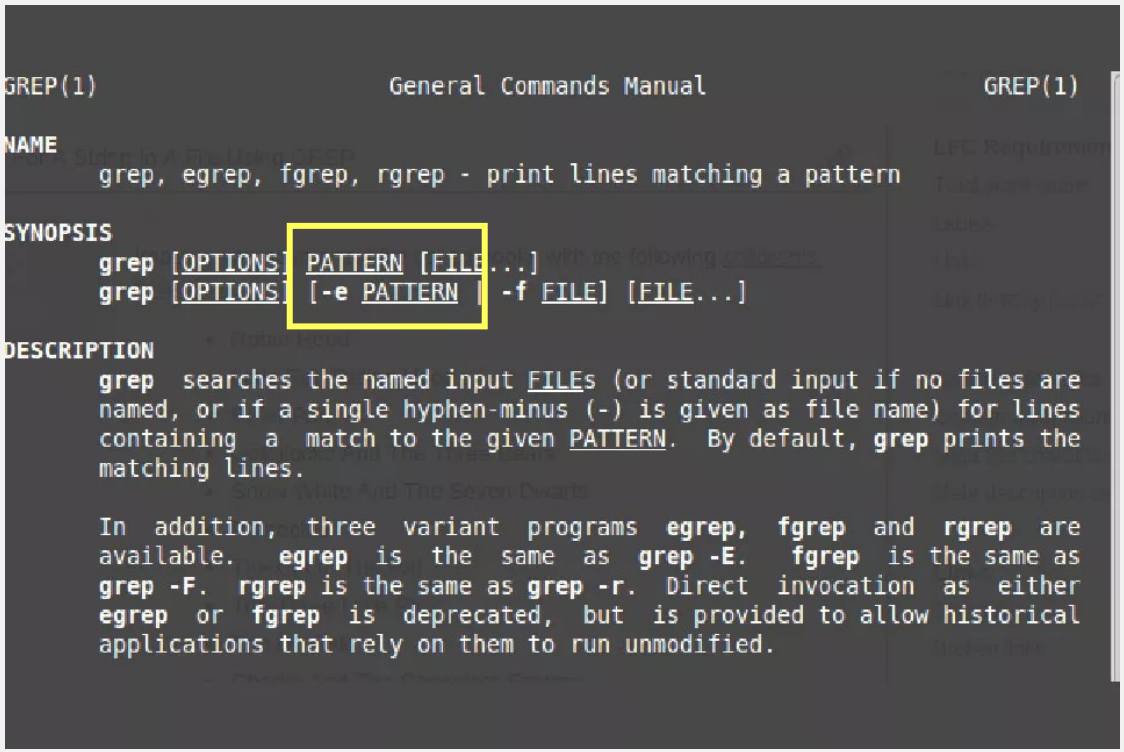
Regular Expressions are Super Useful

- Visual Studio



Regular Expressions are Super Useful

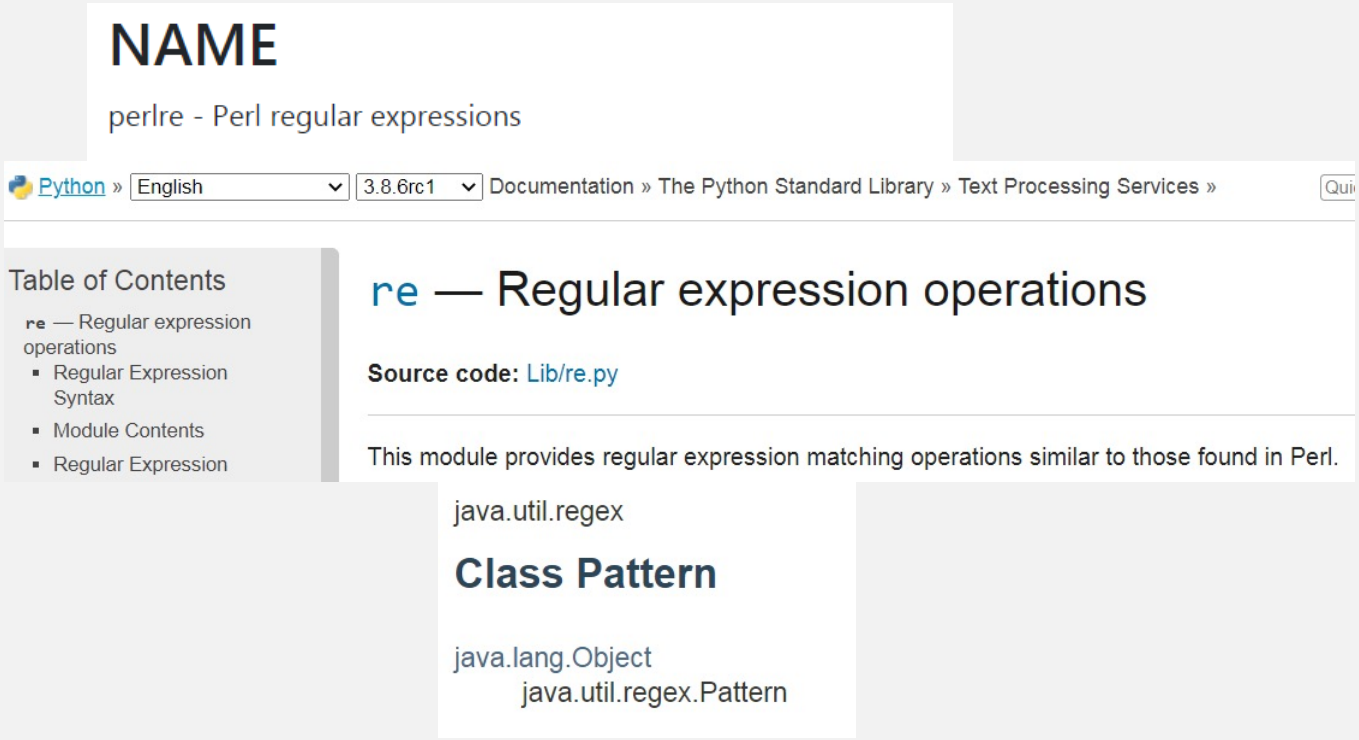
- Grep (Linux)



```
GREP(1)                    General Commands Manual                   GREP(1)

NAME
       grep, egrep, fgrep, rgrep - print lines matching a pattern

SYNOPSIS
       grep [OPTIONS] PATTERN [FILE...]
       grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION
       grep  searches the named input FILEs (or standard input if no files are
       named, or if a single hyphen-minus (-) is given as file name) for lines
       containing  a  match to the given PATTERN.  By default, grep prints the
       matching lines.

       In  addition,  three  variant  programs  egrep,  fgrep  and  rgrep  are
       available.   egrep  is  the  same  as  grep -E.   fgrep  is the same as
       grep -F.  rgrep is the same as grep -r.  Direct  invocation  as  either
       egrep  or  fgrep  is  deprecated,  but  is provided to allow historical
       applications that rely on them to run unmodified.
```

Regexps supported in every language

- Perl
- Python
- Java
- Every lang!



NAME

perlre - Perl regular expressions

Python » English 3.8.6rc1 Documentation » The Python Standard Library » Text Processing Services »

Table of Contents

re — Regular expression operations

- Regular Expression Syntax
- Module Contents
- Regular Expression

re — Regular expression operations

Source code: Lib/re.py

This module provides regular expression matching operations similar to those found in Perl.

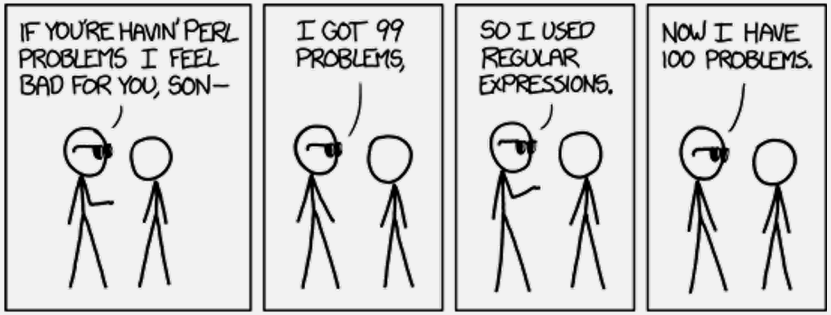
java.util.regex

Class Pattern

java.lang.Object
java.util.regex.Pattern

Caveat: Regexps are useful, if used correctly

Regexps: potentially **useful** …



… only if used correctly

Where "used correctly" = only use it to recognize regular languages

(To do this, you must know what is, and is not, a regular language!)

Someone Who Did Not T

RegEx match open tags except XHTML self-con

Asked 10 years, 10 months ago Active 1 month ago Viewed 2.9m times

1553

I need to match all of these opening tags:

```
<p>
<a href="foo">
```

6572

But not these:

4414

You can't parse [X]HTML with regex. Because HTML can't be parse
Regex is not a tool that can be used to correctly parse HTML. As I h
HTML-and-regex questions here so many times before, the use of re
allow you to consume HTML. Regular expressions are a tool that is
sophisticated to understand the constructs employed by HTML. HTM
regular language and hence cannot be parsed by regular expressio
queries are not equipped to break down HTML into its meaningful pa
times but it is not getting to me. Even enhanced irregular regular exp
used by Perl are not up to the task of parsing HTML. You will never

HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow *it is too late it is too late we cannot be saved* the trangession of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) *dear lord help us how can anyone survive this scourge* using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes *using rege*x as a tool to process HTML establishes a brea*ch between this world* and the dread realm of corrupt entities (like SGML entities, but *more corrupt) a mere glimp*se of the world of reg**ex parsers for HTML will ins**tantly transport a p*rogrammer's consciousness i*nto a wor*l*d of ceaseless screaming, he comes~~, the pestilent sli~~thy regex-infection wi**ll devour your HT**ML parser, application and existence for all time like Visual Basic only worse *he comes he come*s *do not fi*ght h**e com~~es~~, hi**s unholy radiance *destroying all enlightenment, HTML tags **leaking from your eyes like liq**uid p*ain, the song of regular expre~~ssion parsing~~ will ext*inguish the voices of mor**tal man from the sp**here I can see it can you see it it is beautiful* the f`inal snuf`fing *of the lie**s of Man ALL IS LOST ALL IS LO**ST the pony he come*s he co~~mes he comes~~ **the ich**or permeates *all* MY FAC*E MY FACE oh god no* **NOOOOOOO N**O stop *the an*gles `are` **not real ZALGO IS TONY THE PONY HE COMES**

Have you tried using an XML parser instead?

First Section of Course Road Map

- We ultimately want to prove:
  - Regular Languages ⇔ Regular Expressions

Next!

- First, we need to show these operations are closed for reglangs:
  - Union (**done**!)
  - Concatentation (**done**!)
  - Kleene star (**done**!)

Check-in Quiz

On gradescope