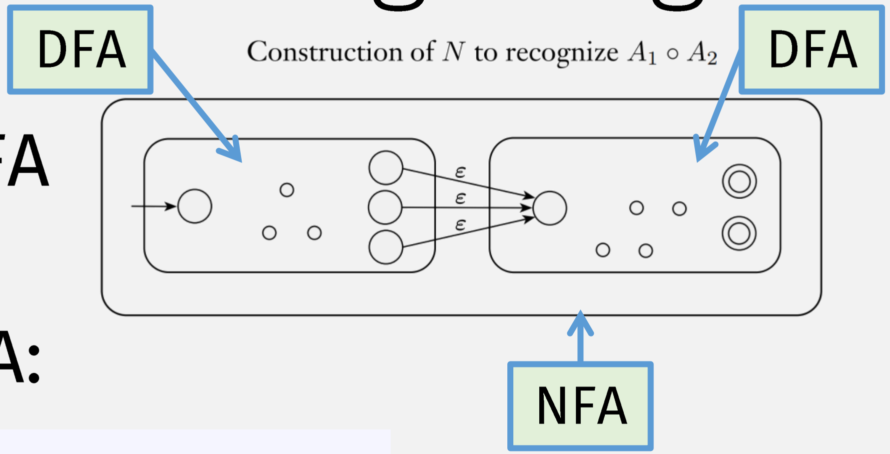
NFA -> DFA
and
Intro to Regular Expressions

Wed, February 10, 2021

Logistics

- Reminder: no class next Monday 2/15/2021
- Welcome new TA: Benjamin Kwapong
  - See website for additional office hour times
- HW1: solutions posted (to piazza)
  - May use ideas, but not copy (obvi)
- HW2: due Sunday 2/14 11:59pm EST
- HW3: posted, due Sunday 2/21 11:59pm EST
  - Includes a non-coding question
- **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 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
    - To complete this part of proof: convert DFA to an NFA! (how?)
  - **<=** 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

How to convert NFA -> DFA?

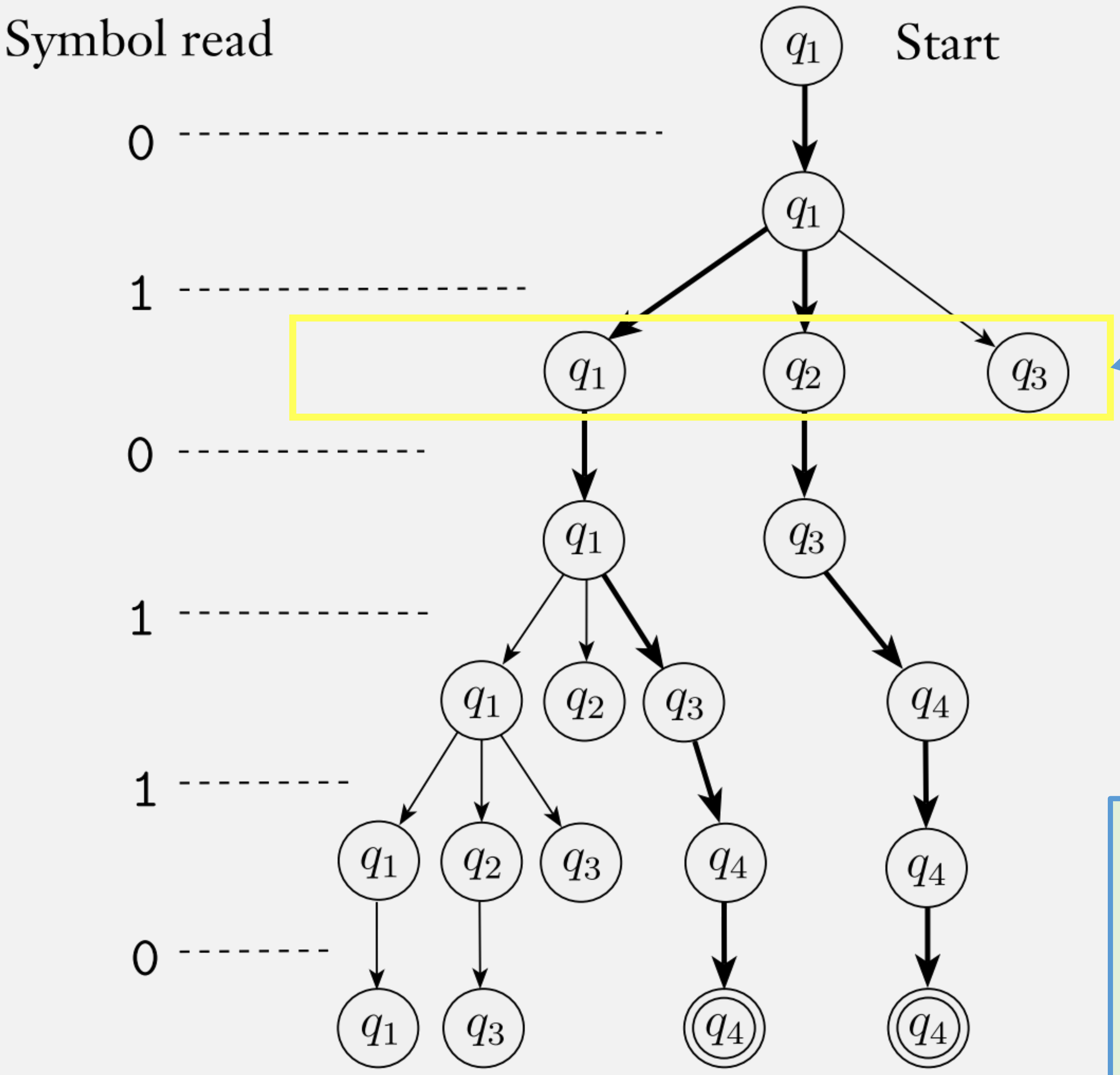
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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



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”

Convert NFA -> DFA, Formally

- Let NFA N = $(Q, \Sigma, \delta, q_0, F)$
- An equivalent DFA M has states Q' = $\mathcal{P}(Q)$ (power set of Q)

Example:

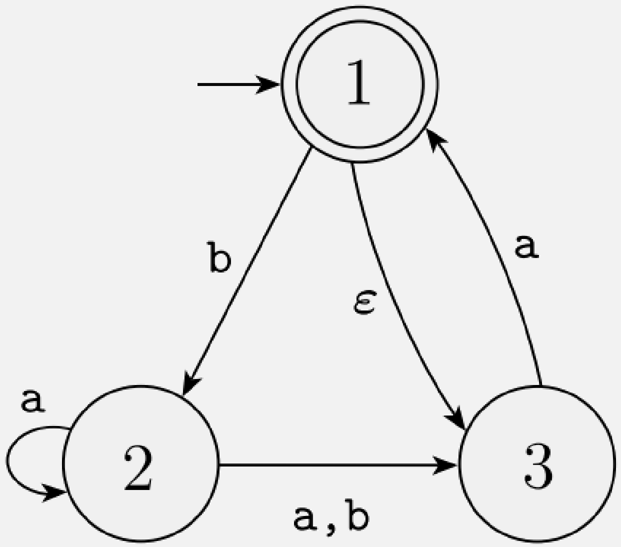


FIGURE 1.42
The NFA N_4

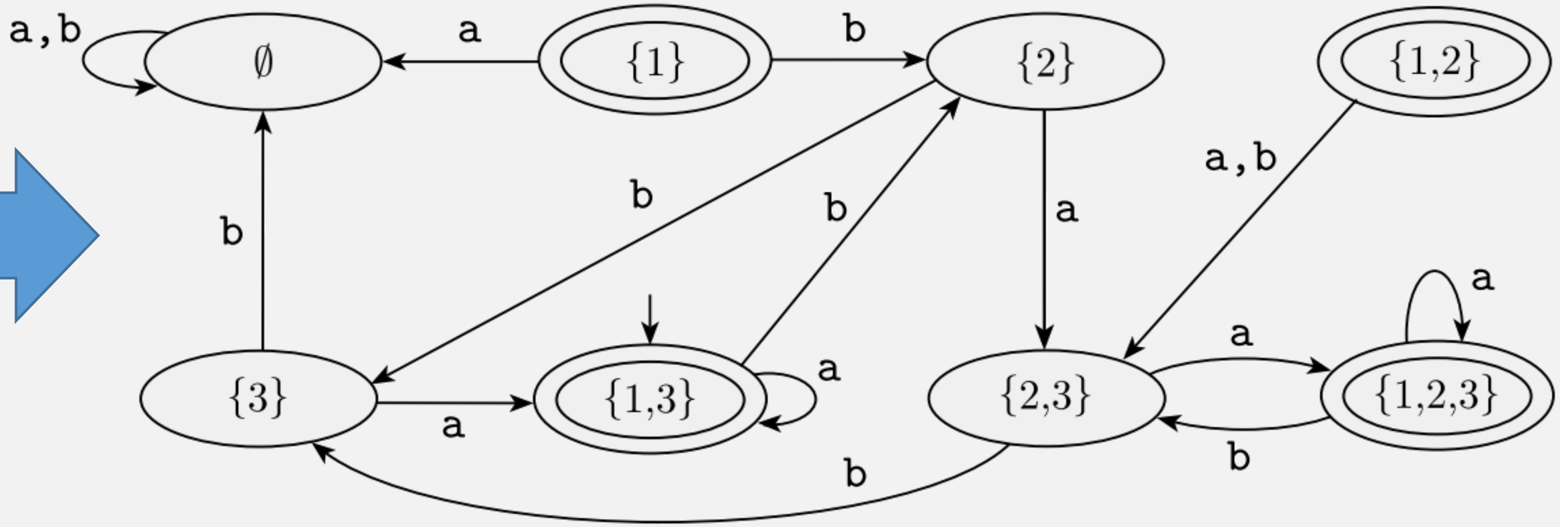


FIGURE 1.43
A DFA D that is equivalent to the NFA N_4

NFA -> DFA (ignore empty transitions)

- <u>Have</u>: $N = (Q, \Sigma, \delta, q_0, F)$
- <u>Want to</u>: 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. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

NFA -> DFA (with empty transitions)

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

$$\boxed{E(R) = \{q \mid q \text{ can be reached from } R \text{ by traveling along 0 or more } \varepsilon \text{ arrows}\}}$$

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

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

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

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

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

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

Proving that NFAs Recognize Reg Langs

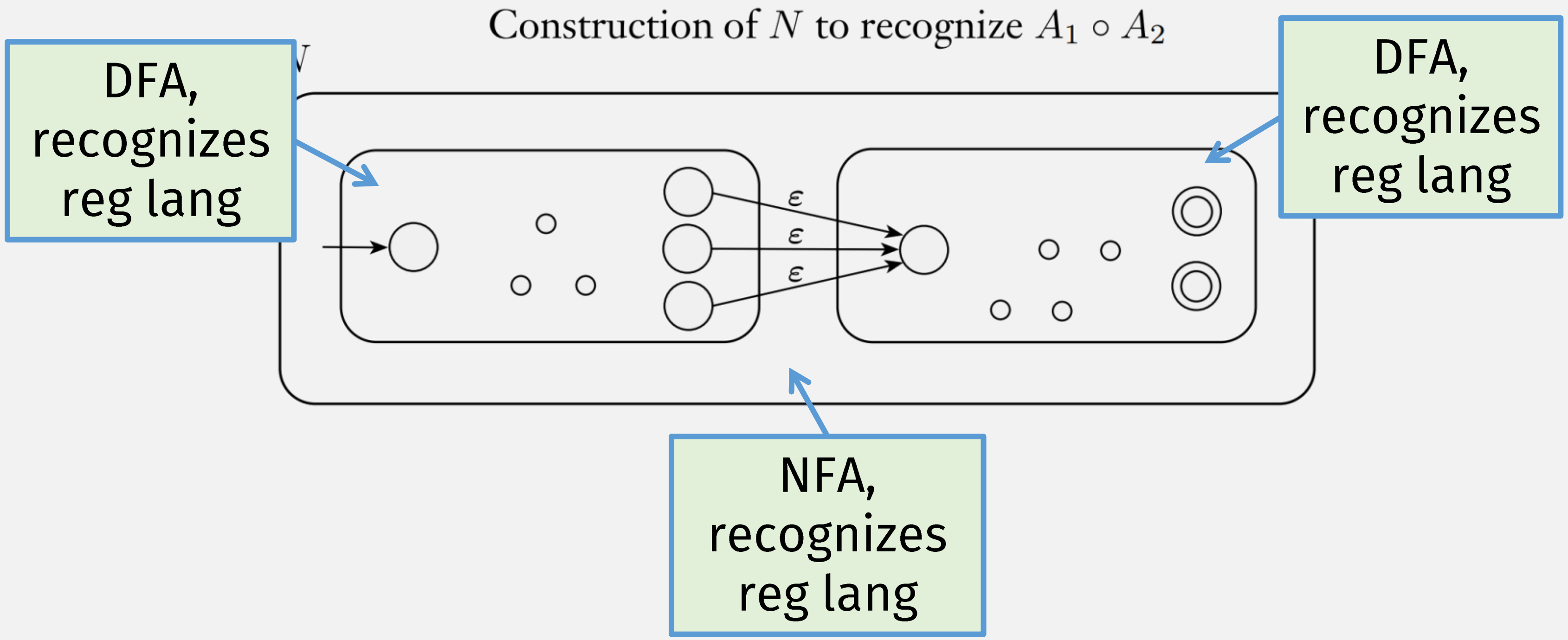
- <u>*Theorem*</u>:
  - A language A is regular **if and only if** some NFA N recognizes it.

- Must prove:
  - **=>** If A is regular, then some NFA N recognizes it
    - Easy
    - <u>We know</u>: if A is regular, then a <u>DFA</u> recognizes it
    - Convert DFA to an NFA
  - **<=** If an NFA *N* recognizes A, then A is regular
    - Hard
    - <u>We know</u>: if a DFA recognizes a lang, then it is regular
    - <u>Idea</u>: 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

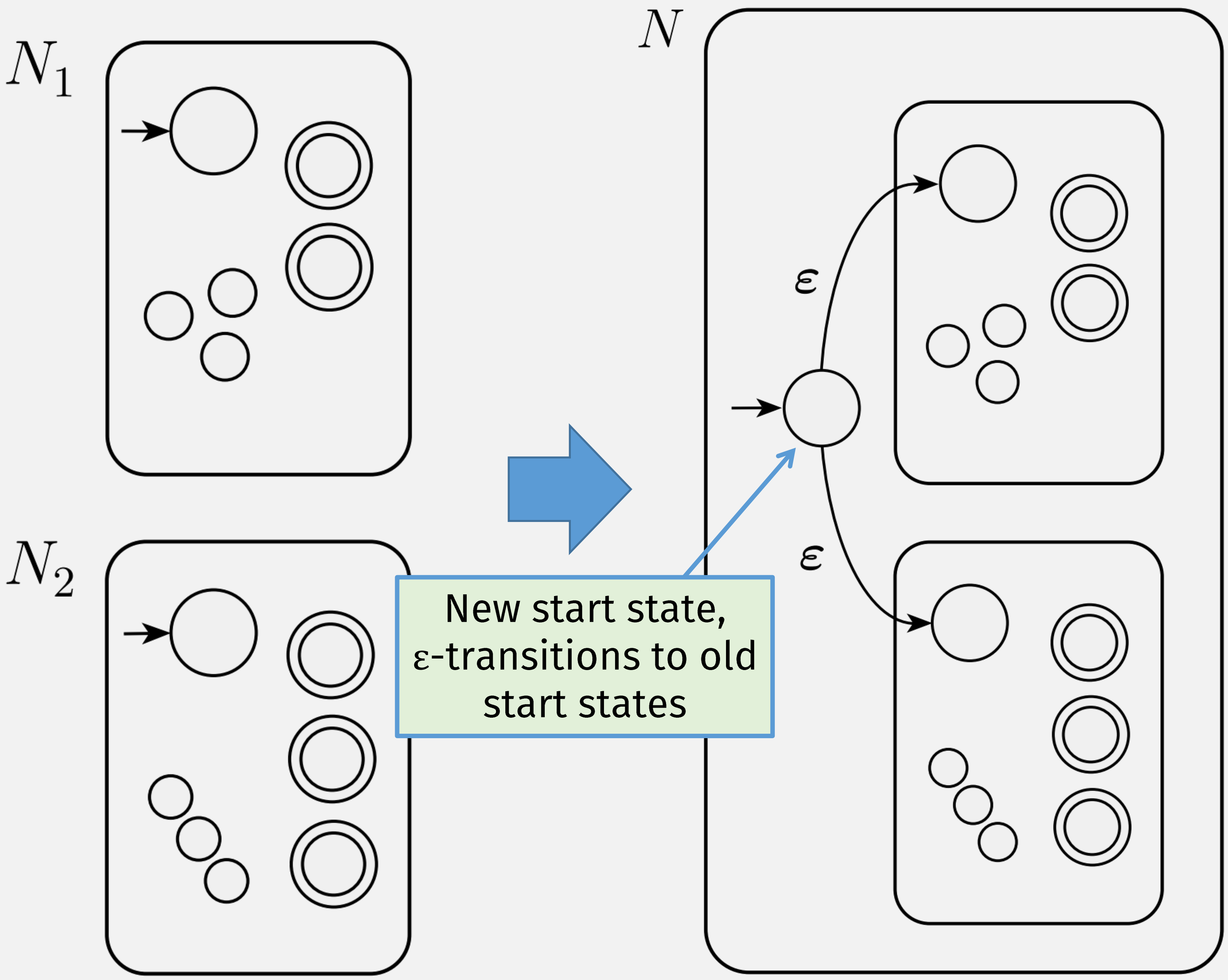


“Regular” Operations

- 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

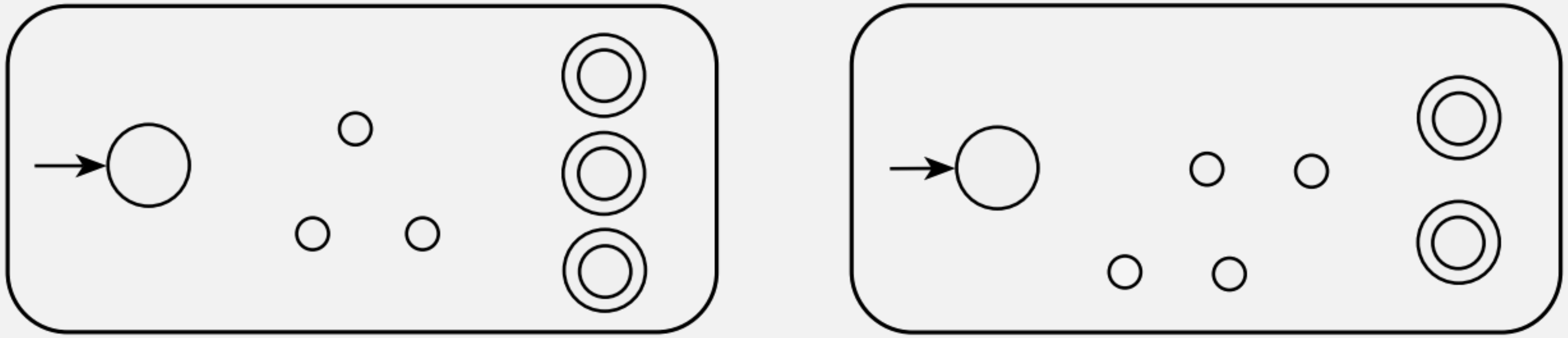
Union



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

Concat

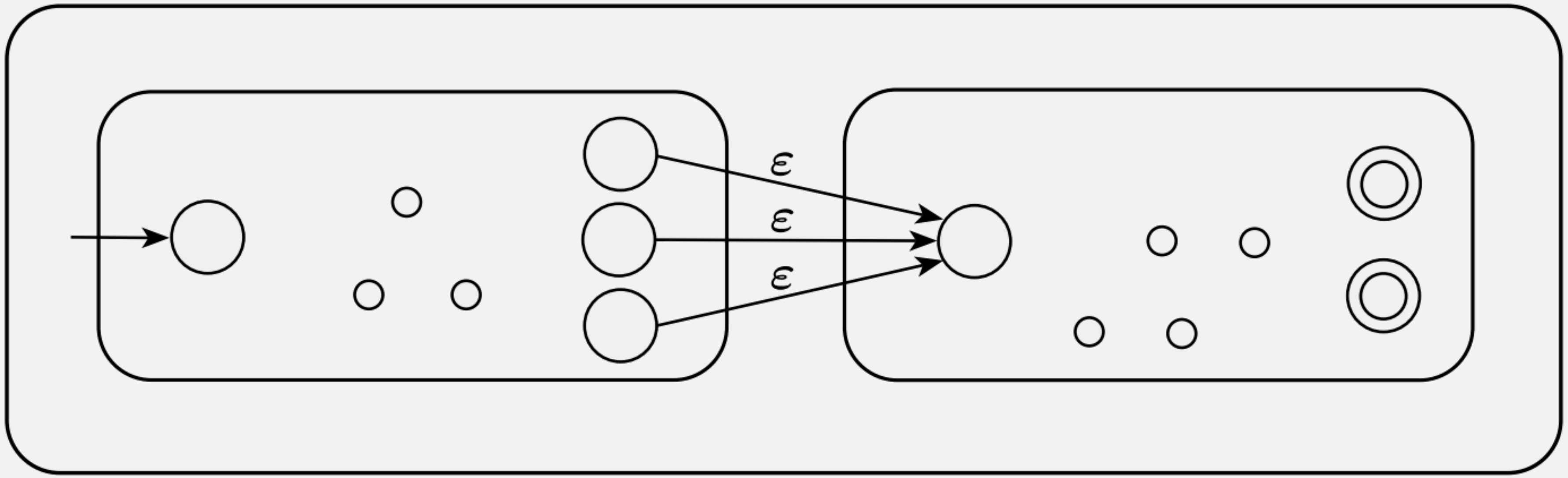
N_1 N_2



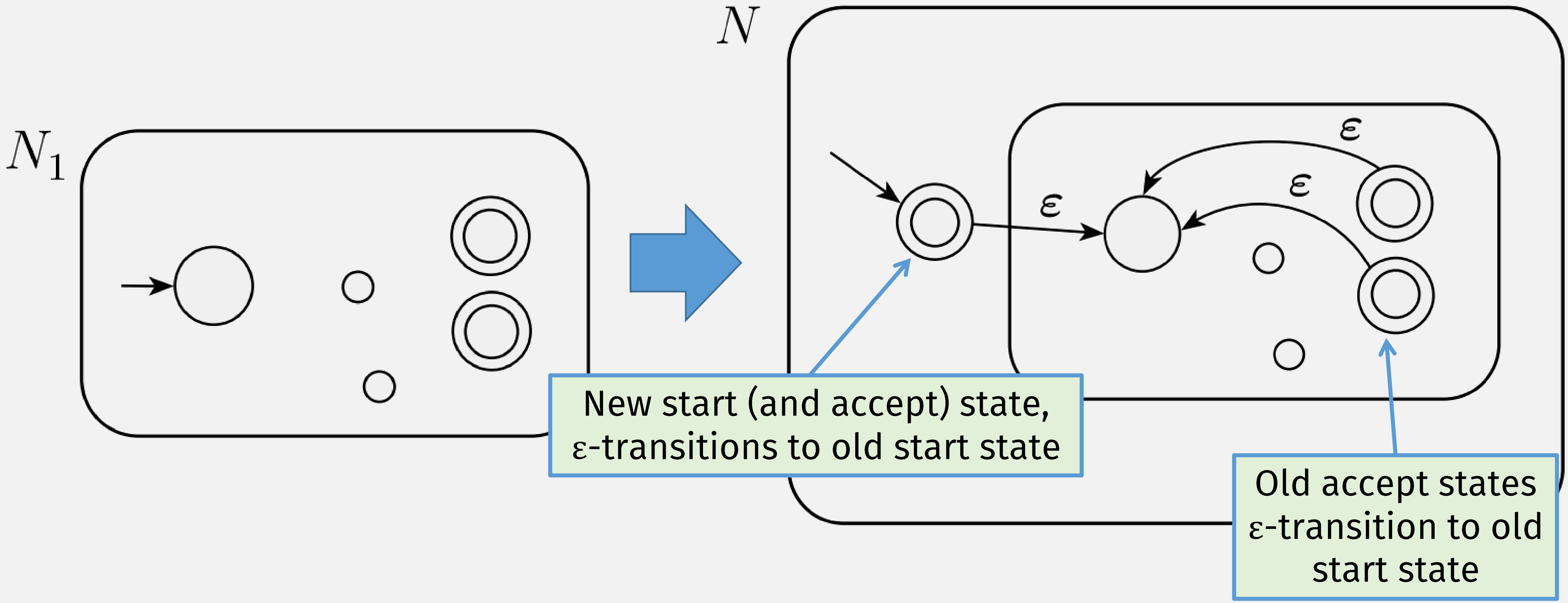
Let N_1 recognize A_1, and N_2 recognize A_2.

Construction of N to recognize $A_1 \circ A_2$

N



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 <u>all regular languages</u>
- 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

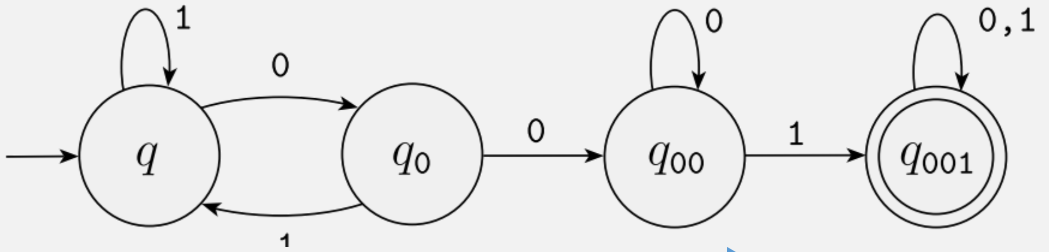
concat

star

Poll: Regexes

Ways to Recognize a Regular Language

- Instead of:



- Or:

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

	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. $F = \{q_2\}$.

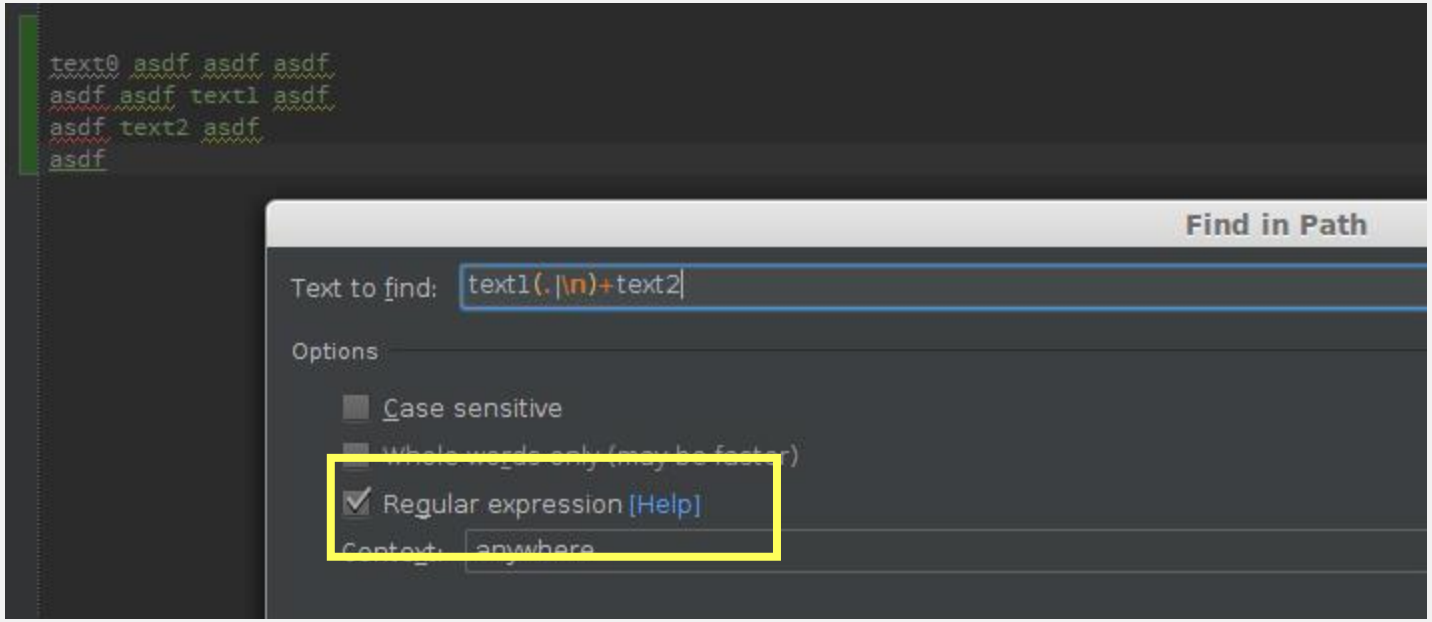
- We can write a regexp: $\Sigma^* 001 \Sigma^*$

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

Which would you rather write?

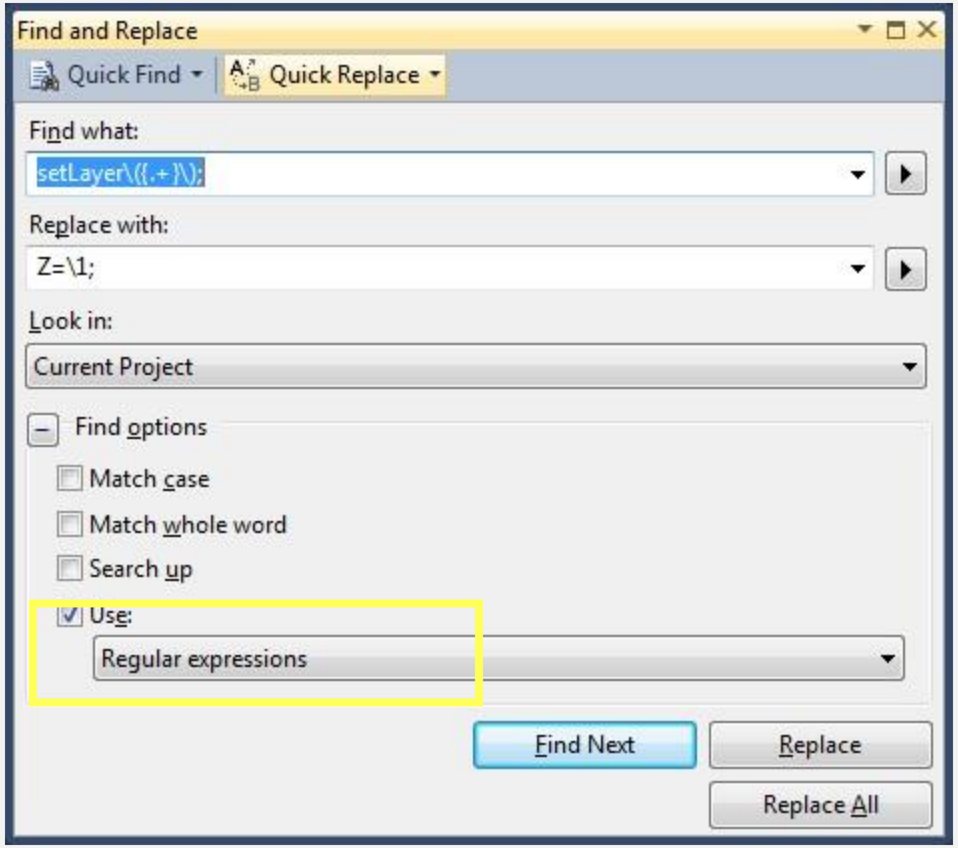
Regular Expressions are Super Useful

- IntelliJ



Regular Expressions are Super Useful

- Visual Studio



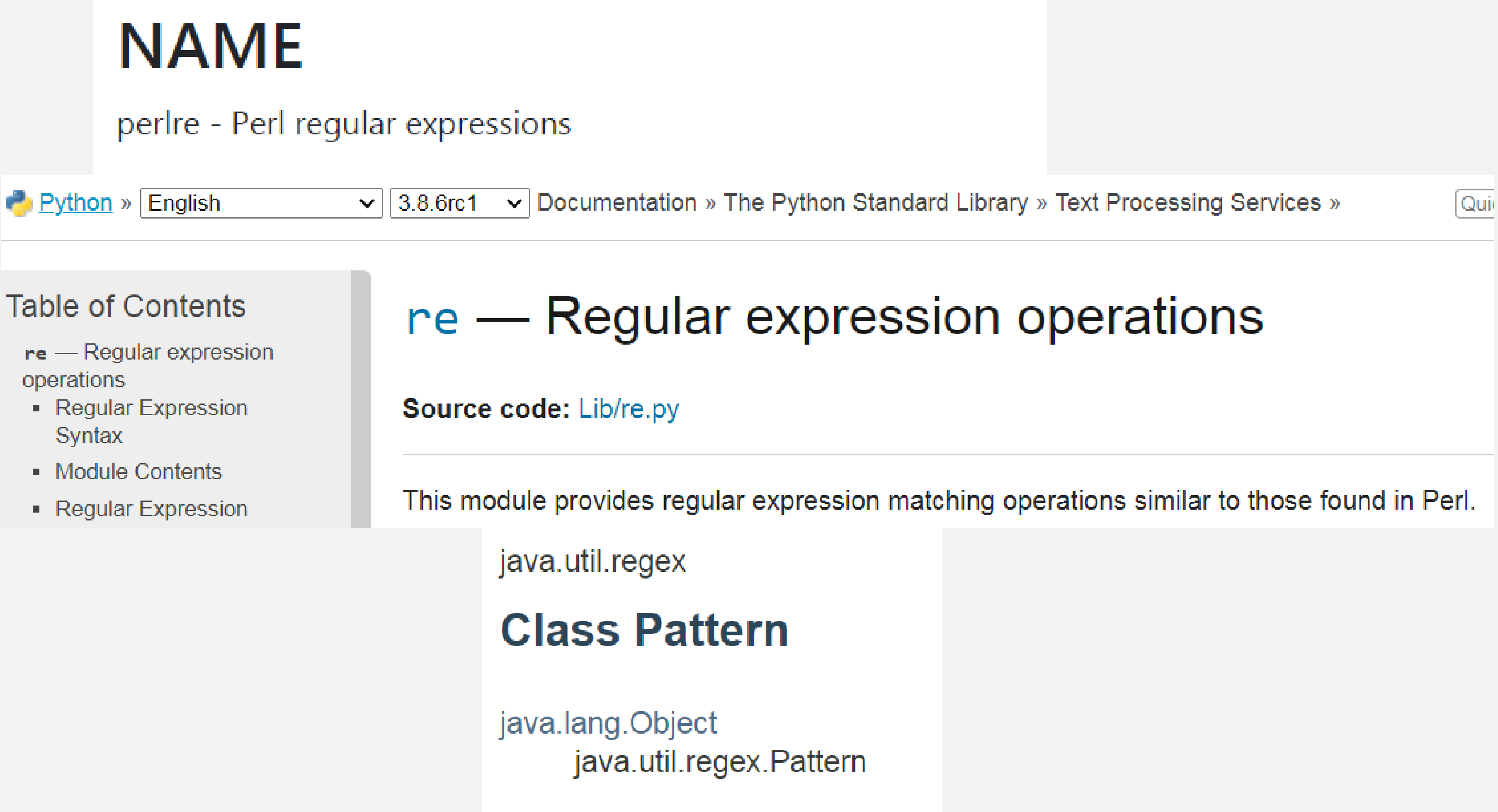
Regular Expressions are Super Useful

- 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.
```

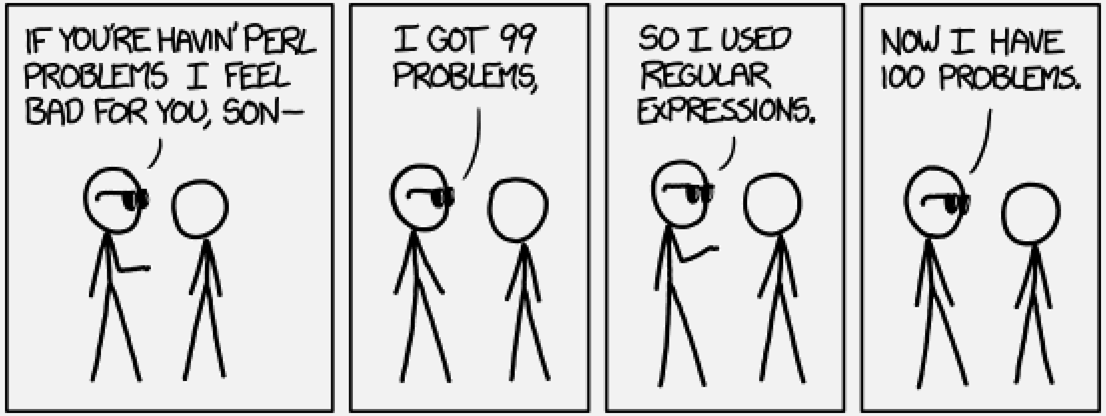
Regexps supported in every language

- Perl
- Python
- Java
- Every lang!



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

6572

I need to match all of these opening tags:

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

But not these:

4414

You can't parse [X]HTML with regex. Because HTML can't be parsed
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 <center> 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 regex* 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 rege**x parsers for HTML will ins**tantly transport a p*rogrammer's consciousness i*nto a w*orl*d of ceaseless screaming, he comes~~, the pestilent s~~lithy regex-infection wil**l devour your HT**ML parser, application and existence for all time like Visual Basic only worse *he comes he comes do not fi*ght h**e com̡e̶s, ̕h**i̧s un̨ho͞ly radiańcé de*stro҉ying all enli̍̈́̂̈́ghtenment, HTML tags* **lea͠ki̧n͘g fr̶ǫm ̡yo͟ur eye͢s̸ ̛l̕ik͏e liq**uid pain, the song of re̸gular expre~~ssion parsing~~ will exti*nguish the voices of mor*tal **man from the sp**here *I can see it can you see ̲͚̖͔̙î̩́t̲͎̩̱͔́̋̀ it is beautiful t*he f`inal snuf`fing *o*f the lie*s of Man ALL IS LOŚ͖̩͇̗̪̏̈́T ALL I*S LOST th̷e pon̷y he come*s he c̶̮omes ~~he comes~~ t**he ich**or permeates all MY FACE MY FACE ᵒh god no NO NOOOO NΘ stop t*he an*̶͑̾̾̅ͫ͏̙̤g͇̫͛͆̾ͫ̑͆l͖͉̗̩̳̟̍ͫͥͨe̠̅s `are` n̳̳̞̘̊͌̑̚ot rè̑ͧ̌aͨl̘̝̙̃ͤ͂̾̆ ZA̡͊͠͝LGΌ ISͮ̂҉̯͈͕̹̘̱ TO͇̹̺ͅƝ̴ȳ̳ TH̘Ë͖́̉ ͠P̯͍̭O̚N̐Y̡ H̸̡̪̯ͨ͊̽̅̾̎Ȩ̬̩̾͛ͪ̈́̀́͘ ̶̧̨̱̹̭̯ͧ̾ͬC̷̙̲̝͖ͭ̏ͥͮ͟Oͮ͏̮̪̝͍M̲̖͊̒ͪͩͬ̚̚͜Ȇ̴̟̟͙̞ͩ͌͝S̨̥̫͎̭ͯ̿̔̀ͅ

Have you tried using an XML parser instead?

Big Picture 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**!)

<u>Thm</u>: A lang is regular iff some regexp describes it

- => If a language is regular, it is described by a regexp

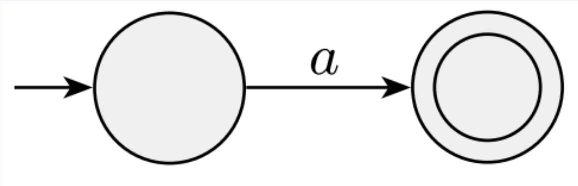
- <= If a language is described by a regexp, it is regular
  - Easy!
  - Construct the NFA!
  - See Lemma 1.55

Regexp -> NFA

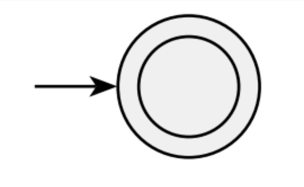
DEFINITION 1.52

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

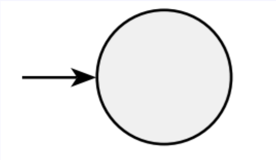
1. a for some a in the alphabet Σ,



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 … r

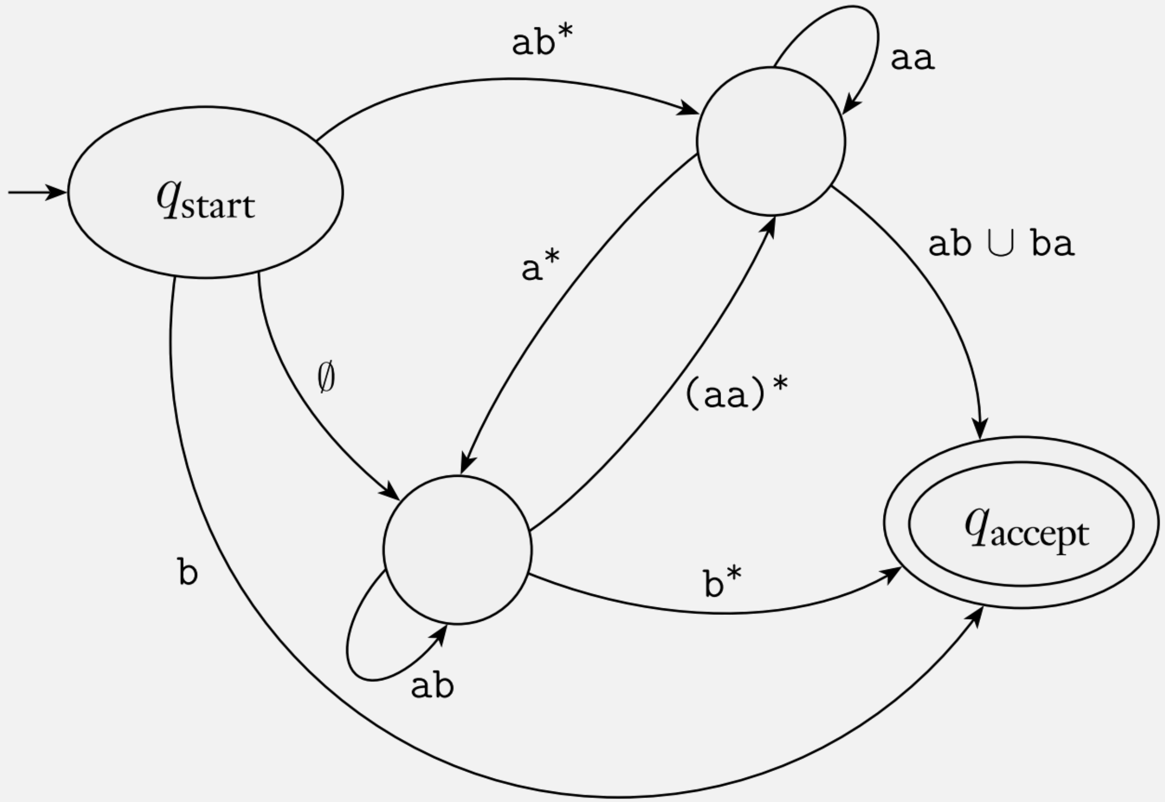
6. (R_1^*), where R_1 is a regular expression.

Constructions from before!

Thm: A lang is regular iff some regexp describes it

- => If a language is regular, it is described by a regexp
  - Hard!
  - Need something new: a GNFA
- <= If a language is described by a regexp, it is regular
  - Easy!
  - Construct the NFA! (**Done**)

GNFA = NFA with regexp transitions



- To <u>convert</u> to regexp, keep "ripping out" states until only 2 are left

Check-in Quiz 2/10

On gradescope