CS 172: Computability and Complexity

Regular Expressions

Sanjit A. Seshia
EECS, UC Berkeley

Acknowledgments: L.von Ahn, L. Blum, M. Blum

The Picture So Far

$$
\begin{tikzcd}
\textbf{DFA} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & \textbf{NFA} \\
\textbf{Regular language} &
\end{tikzcd}
$$

Today's Lecture

$$
\begin{array}{ccc}
\textbf{DFA} & \longleftrightarrow & \textbf{NFA} \\
\updownarrow & & \updownarrow \\
\textbf{Regular language} & & \textbf{Regular expression}
\end{array}
$$

Regular Expressions

- What is a regular expression?

Regular Expressions

- Q. What is a regular expression?
- A. It's a "textual"/ "algebraic" representation of a regular language
  - A DFA can be viewed as a "pictorial" / "explicit" representation

- We will *prove* that a regular expressions (regexps) indeed represent regular languages

Regular Expressions: Definition

σ is a regular expression representing $\{\sigma\}$ ($\sigma \in \Sigma$)

ε is a regular expression representing $\{\varepsilon\}$

$\emptyset$ is a regular expression representing $\emptyset$

If R_1 and R_2 are regular expressions representing L_1 and L_2 then:

(R_1R_2) represents $L_1 \cdot L_2$

$(R_1 \cup R_2)$ represents $L_1 \cup L_2$

$(R_1)^*$ represents L_1^*

Operator Precedence

1. $*$

2. $\cdot$ (often left out; $a \cdot b \rightarrow ab$)

3. $\cup$

Example of Precedence

$$\mathbf{R_1{}^*R_2 \cup R_3 = ((R_1{}^*)R_2) \cup R_3}$$

What’s the regexp?

{ w | w has exactly a single 1 }

0*10*

What language does $\emptyset^*$ represent?

{ε}

What's the regexp?

{ w | w has length ≥ 3 and its 3rd symbol is 0 }

$$\Sigma^2 \, \mathbf{0} \, \Sigma^*$$

$$\Sigma = (0 \cup 1)$$

Some Identities

Let R, S, T be regular expressions

- $R \cup \emptyset = ?$
- $R \cdot \emptyset = ?$
- Prove: $R (S \cup T) = R S \cup R T$

  (what's the proof idea?)

Some Applications of Regular Expressions

- String matching & searching
  – Utilities like grep, awk, …
  – Search in editors: emacs, …
- Programming Languages
  – Perl
  – Compiler design: lex/yacc
- Computer Security
  – Virus signatures

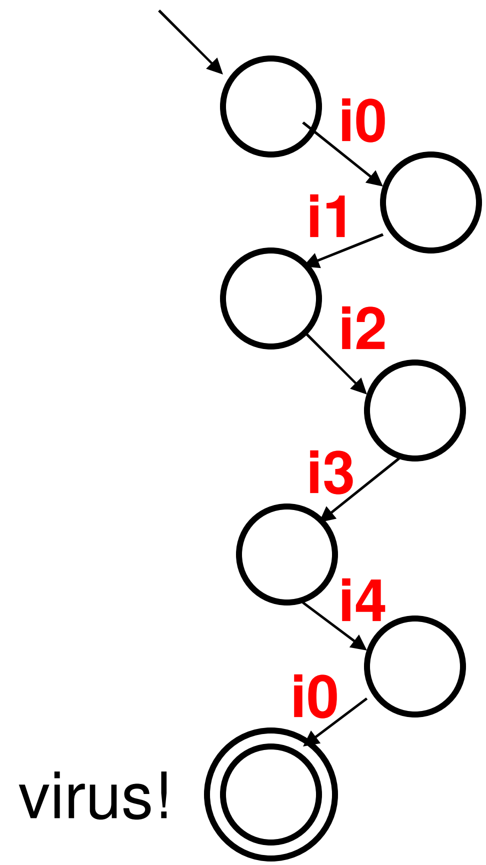
Virus Signature as String

```
…
pop ecx
jecxz SFModMark
mov esi, ecx
mov eax, 0d601h
pop edx
pop ecx
…
```

Chernobyl virus
code fragment

Sequence of words, one for each instruction:

i0
i1
i2
i3
i4
i0



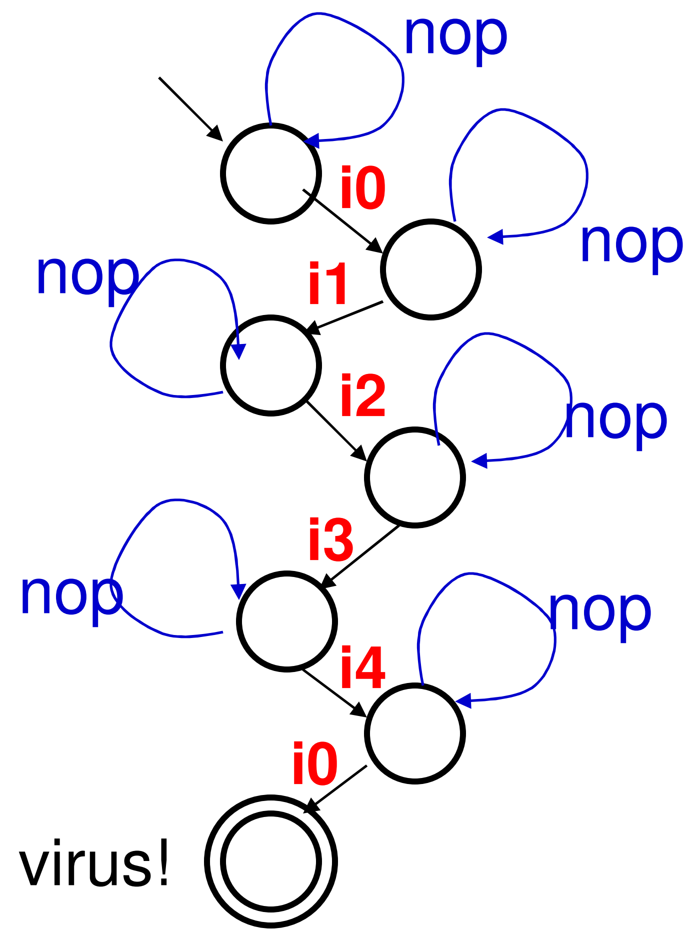
Virus Signature as Regexp

```
…
nop
pop ecx
nop
jecxz SFModMark
mov esi, ecx
nop
nop
mov eax, 0d601h
pop edx
pop ecx
…
```

Simple obfuscated Chernobyl virus code fragment

Sequence of words doesn't work!

i0
i1
i2
i3
i4
i0



Equivalence Theorem

A language is regular

$\Uparrow$ if and only if $\Downarrow$

some regular expression describes it

Part I (“if part”)

Some regular expression R
describes a language

$\Rightarrow$

That language is regular

There exists NFA N such that R describes L(N)

Given regular expression R, we show there exists NFA N such that R represents L(N)

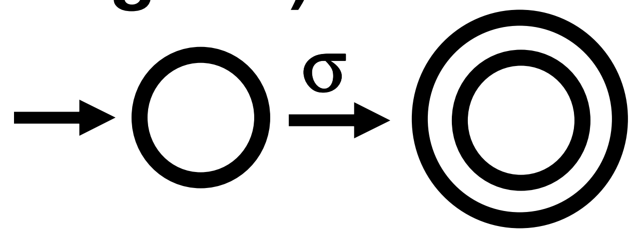
Proof idea?

Given regular expression R, we show there exists NFA N such that R represents L(N)

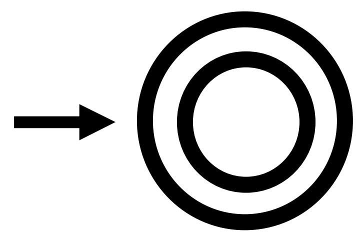
Proof Idea: Induction on the length of R:

Base Cases (R has length 1):

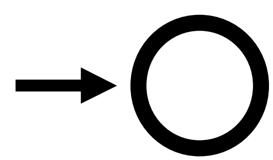
R = σ



R = ε



R = ∅



Inductive Step:

Assume R has length $k > 1$ and that any regular expression of length $< k$ represents a language that can be recognized by an NFA

What might R look like?

$$R = R_1 \cup R_2$$

$$R = R_1 R_2$$

$$R = (R_1)^*$$

(remember: we have NFAs for R_1 and R_2)

Part I ("if part")

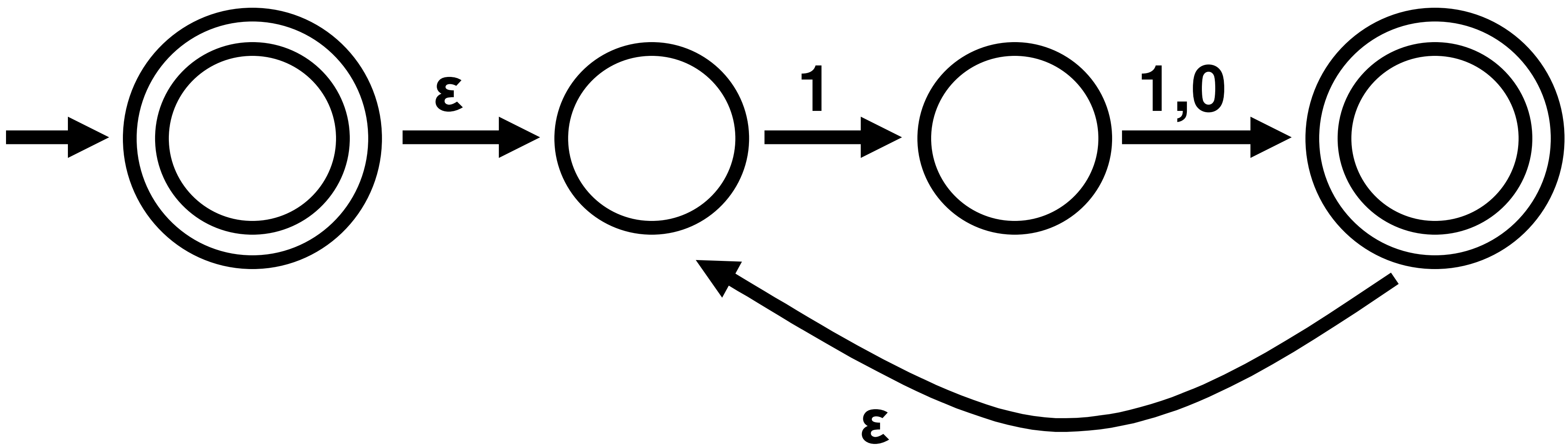
Some regular expression R
describes a language
$\Rightarrow$
That language is regular

There exists NFA N such that R describes L(N)

DONE !

An Example

Transform (1(0 ∪ 1))* to an NFA



Part II (“only if part”)

A language is regular

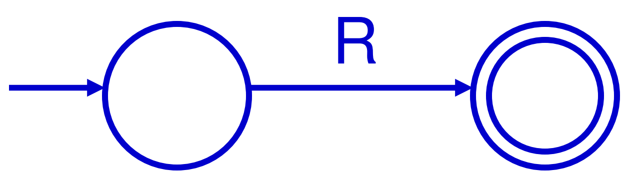
$\Rightarrow$

Some regular expression R describes it

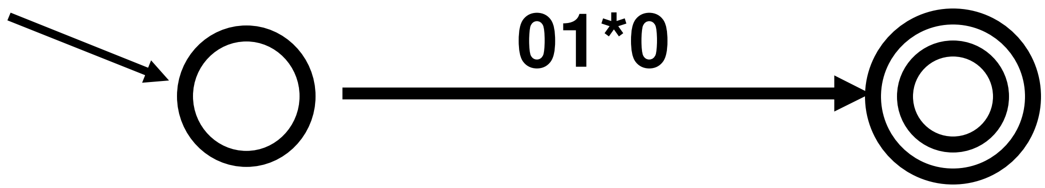
Turn DFA into equivalent regular expression

Proof Sketch

1. DFA → Generalized NFA
   - NFA with edges labeled by regexps, 1 start state, and 1 accept state
2. GNFA with k states → GNFA with 2 states
   - k > 2; delete states but maintain equivalence
3. 2-state GNFA → regular expression R



GNFA Example & Definition



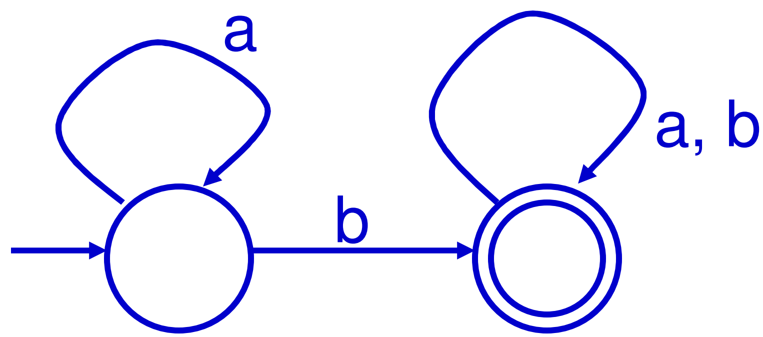
A GNFA is a tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$

- Q – set of states
- Σ – finite alphabet (not regexps)
- q_{start} – initial state (unique, no incoming edges)
  - ε transitions to old start state
- q_{accept} – accepting state (unique, no outgoing edges)
  - ε transitions from old accept states
- $\delta : (Q \setminus q_{accept}) \times (Q \setminus q_{start}) \rightarrow R$

  R – set of all regexps over Σ.

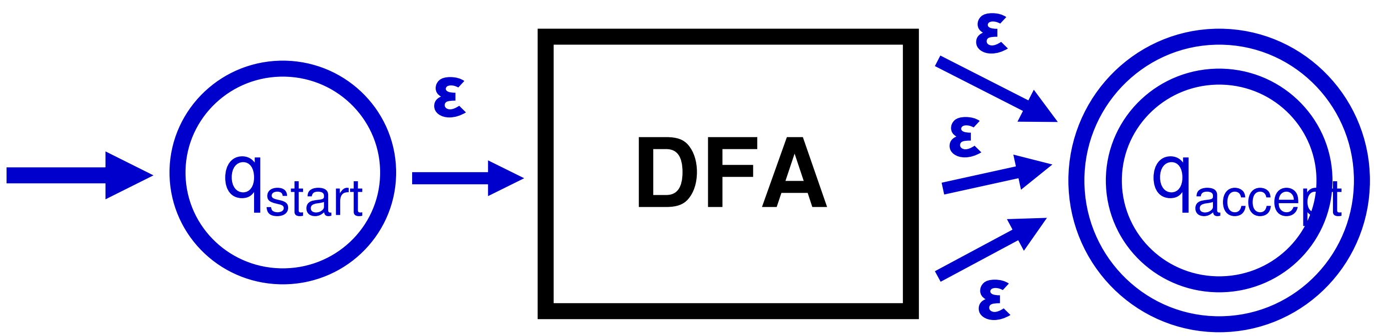
  Example: Any string matching 01^*0 can cause the transition.

Step 1: DFA to GNFA



What's the corresponding GNFA?

Step 1: DFA to GNFA



Add unique and distinct start and accept states

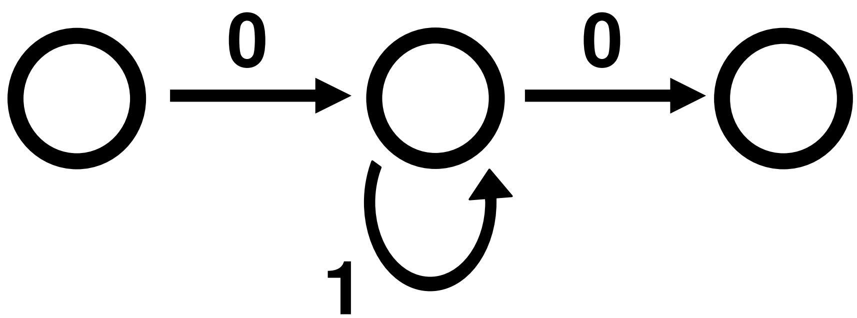
Edges with multiple labels → regexp labels

If internal states (q_1, q_2) don't have an edge between them, add one labeled with $\emptyset$

Step 2: Eliminate states from GNFA

While machine has more than 2 states:

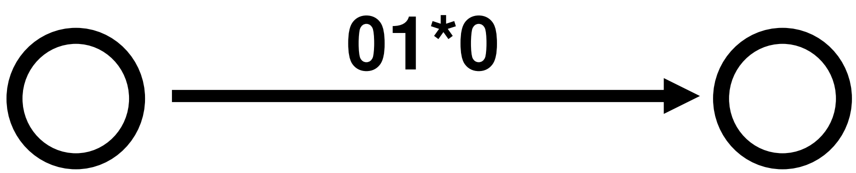
Pick an internal state, rip it out and re-label the arrows with regular expressions to account for the missing state

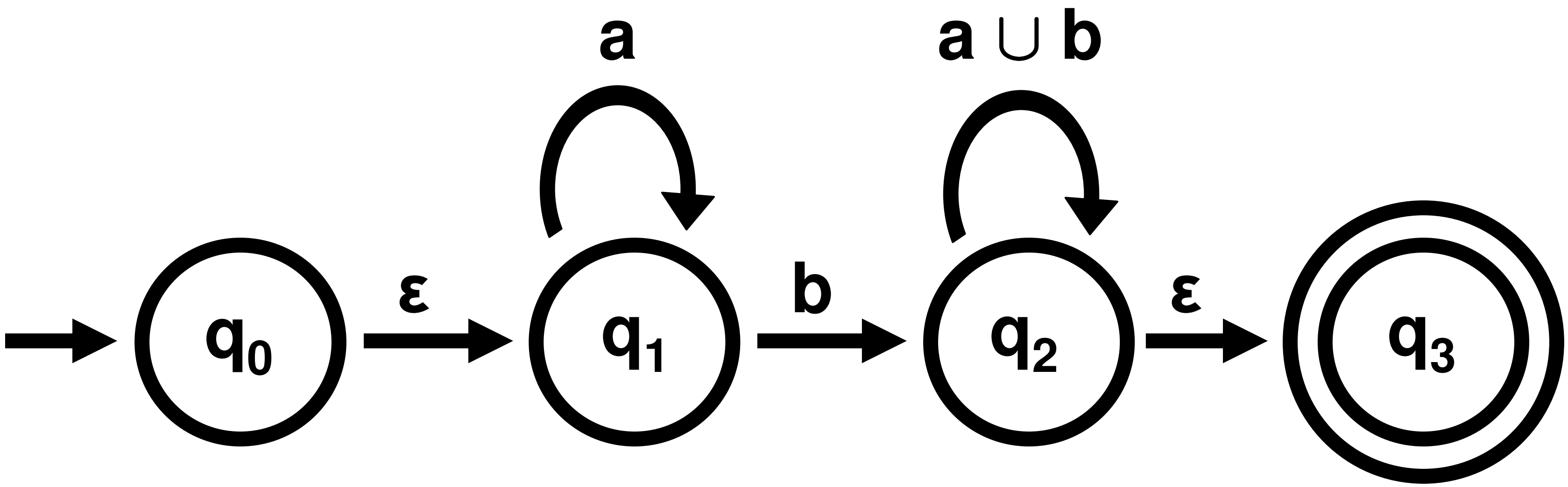


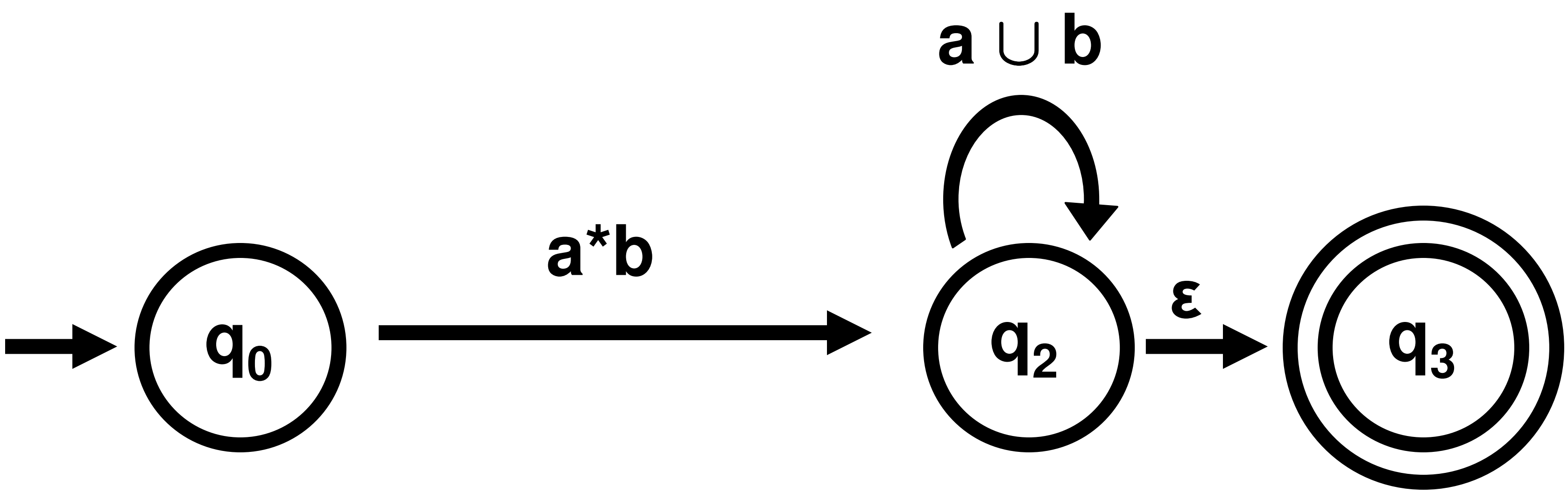
Step 2: Eliminate states from GNFA

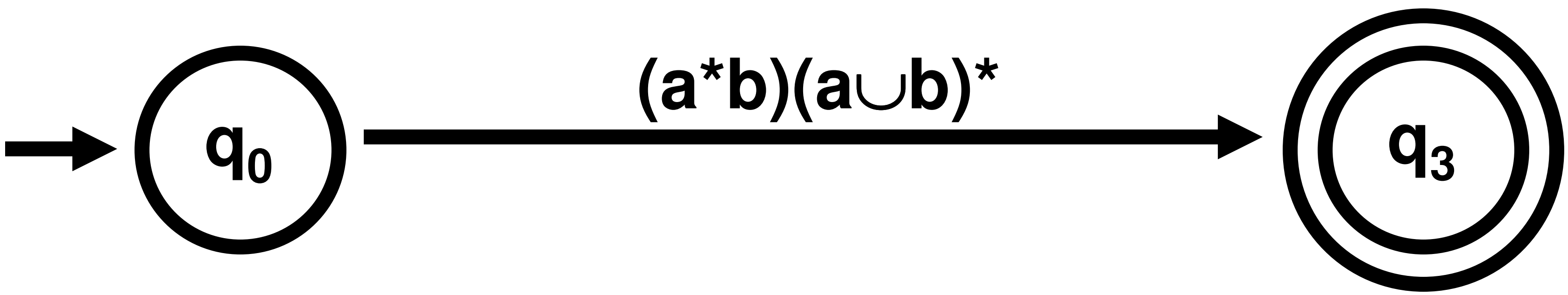
While machine has more than 2 states:

Pick an internal state, rip it out and re-label the arrows with regular expressions to account for the missing state




a
a ∪ b
ε
b
ε
q_0
q_1
q_2
q_3


a ∪ b
a*b
ε
q0
q2
q3



$\delta(q_0,q_3) = (a^*b)(a \cup b)^*$

Formally: Add q_{start} and q_{accept} and create GNFA G

Run CONVERT(G) to eliminate states & get regexp:

If #states = 2

return the expression on the arrow going from q_{start} to q_{accept}

If #states > 2

?

Formally: Add q_{start} and q_{accept} to create G

Run CONVERT(G):

If #states > 2

select $q_{rip} \in Q$ different from q_{start} and q_{accept}

define $Q' = Q - \{q_{rip}\}$

define δ' as:

$$\delta'(q_i,q_j) = \delta(q_i,q_{rip})\delta(q_{rip},q_{rip})^*\delta(q_{rip},q_j) \cup \delta(q_i,q_j)$$

return CONVERT(G′) /* recursion */

(what does this look like, pictorially?)

Prove: CONVERT(G) is equivalent to G

Proof *by induction on k* (number of states in G)

Base Case:

- ✓ **k = 2**

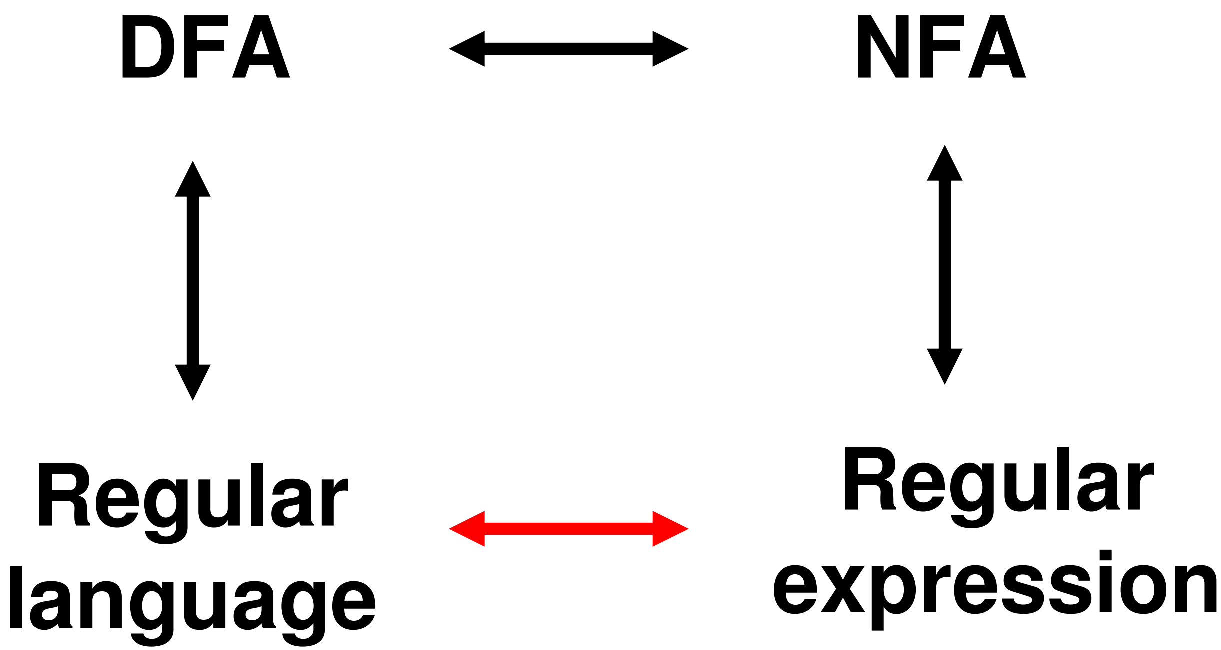
Inductive Step:

Assume claim is true for k-1 states

Prove that G and G′ are equivalent

By the induction hypothesis, G′ is equivalent to CONVERT(G′)

The Complete Picture



Which language is regular?

C = { w | w has equal number of 1s and 0s}

NOT REGULAR

D = { w | w has equal number of occurrences of 01 and 10}

REGULAR!

Next Steps

- Read Sipser 1.4 in preparation for next lecture