CMSC 141

Automata and Language Theory

jmsamaniego@uplb.edu.ph

- Finite automata and regular languages; push-down automata and context-free languages; Turing machines and recursively enumerable sets; linear-bounded automata and context-sensitive languages; computability and the halting problem; undecidable problems; recursive functions and intro to computational complexity (normally offered 1st and 2nd semester)
- 3 units; prerequisite: CMSC 123 (Data Structures) or COI

Course Overview

jmsamaniego@uplb.edu.ph

- We study formal models of computation
- Including their construction, limitations, mathematical and computational properties, equivalence with other models, etc.

Selected references and tools

jmsamaniego@uplb.edu.ph

- M. Sipser. Introduction to the Theory of Computation. Thomson, 2007.
- J.E. Hopcroft, R. Motwani and J.D. Ullman. Introduction to Automata Theory, Languages and Computation. 2nd ed, Addison-Wesley, 2001.
- H.R. Lewis, C.H. Papadimitriou. Elements of the theory of computation, Prentice-Hall, 2nd ed, 1998.
- D.I.A. Cohen. Introduction to Computer Theory. 2nd ed, Wiley, 1997.
- M.D. Davis, R. Sigal, E.J. Weyuker. Computability, Complexity and Languages: Fundamentals of Theoretical Computer Science, 2nd ed, Morgan Kaufmann, 1994.
- E.A. Albacea. Automata, Formal Languages and Computations, UPLB Foundation, Inc. 2005.
- R. Sedgewick and K. Wayne, Theory of Computation, http://www.cs.princeton.edu/introcs/70theory/
- JFLAP, www.jflap.org
- Kara, http://www.infsec.ethz.ch/education/ss04/theoinf/
- Lex/Flex and Yacc/Bison, http://dinosaur.compilertools.net/

Key ideas

jmsamaniego@uplb.edu.ph

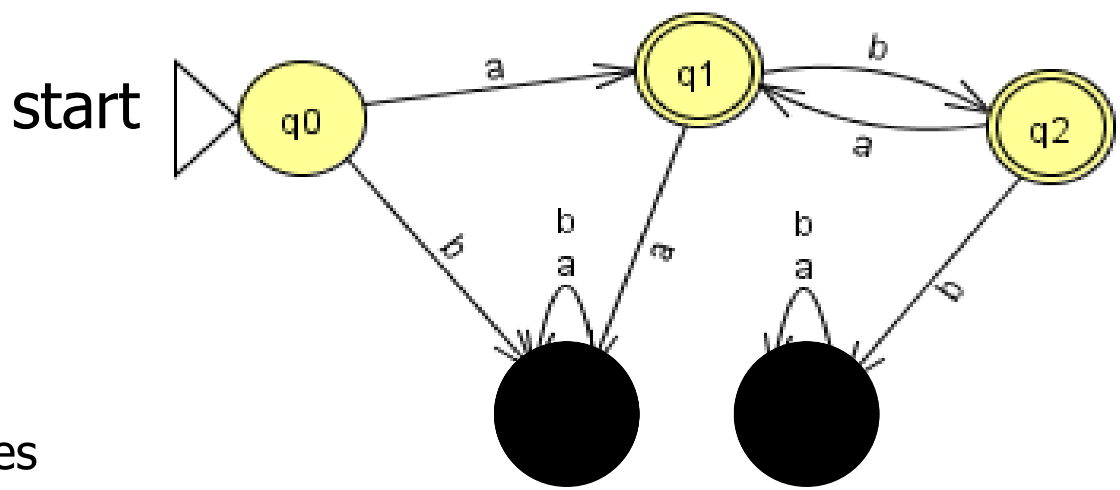
- Automaton = a theoretical machine that
  - Accepts or rejects input strings
  - Performs some string processing (computation)
- Formal language = a set of strings (set can be finite, but most interesting languages are infinite)

A first example

jmsamaniego@uplb.edu.ph

L = { a, ab, aba, abab, ababa, … }

= set of strings *x* over the alphabet Σ = {a, b}, such that *x* starts with *a* and alternates with *b*

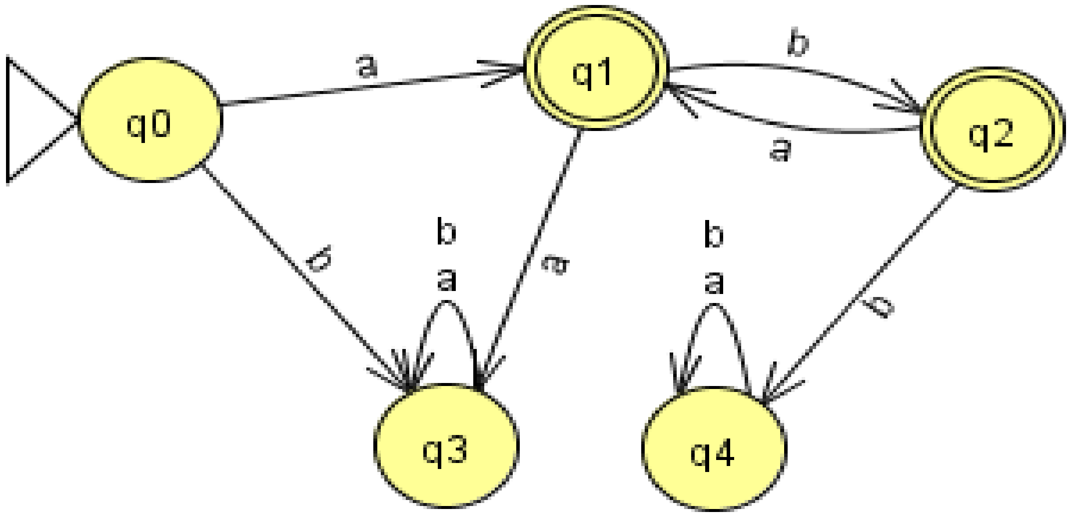


final or accepting states

dead states or blackholes

A finite automaton

jmsamaniego@uplb.edu.ph

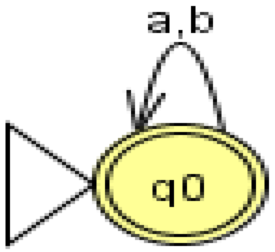


- This finite state machine accepts all strings in L and rejects all others
- The machine has a finite number of nodes, but accepts an infinite number of strings

Exercises

Exercises train the mind … be sure to try the following:

- What does this automaton accept?
- Do we really need two blackholes in the previous slide?
- Is there an equivalent automaton for our example with only three states? Why or why not?
- Design a similar automaton that accepts all alternating strings over {a,b} and can start with either *a* or *b*.



jmsamaniego@uplb.edu.ph

Why study automata theory and formal languages?

- They form the foundations of most branches of computer science
- These topics (like logic and many other branches of math) will be forever classics
- Working with abstract machines and languages train the mind; you will be better programmers and analysts after this course
- Applications in language/compiler design, natural language translation, fractal graphics, bioinformatics, etc.

jmsamaniego@uplb.edu.ph

Alphabets and strings

jmsamaniego@uplb.edu.ph

- An alphabet Σ is a finite set of symbols
  - $\Sigma_1 = \{a, b\}$
  - $\Sigma_2 = \{ 0, 1, 2, \ldots, 9 \}$
  - Σ_3 = ASCII (or UNICODE) character set
- A string is a finite sequence of symbols over some alphabet Σ
  - a, ab, baa are some strings over Σ_1

jmsamaniego@uplb.edu.ph

Strings and string operations

- String concatenation, e.g.
  - if x = abb and y = ab then xy = abbab
- ε represents the empty string
  - For any string x, $x\,\varepsilon = \varepsilon\, x = x$
  - Some books use λ or Λ for the empty string
- We use exponent-notation for self-concat
  - If x = aab then x^2 = aabaab, x^3 = aabaabaab
  - By convention, $x^0 = \varepsilon$ for any string x

More on strings

jmsamaniego@uplb.edu.ph

- x is a *prefix* of w
  y is a *substring* of w
  z is a *suffix* of w

  } if w = xyz

- |x| denotes the length of a string, e.g.,
  |abb| = 3 and |ε| = 0

jmsamaniego@uplb.edu.ph

Exercises

- Is concatenation commutative? Is it associative?
  - Is $xy = yx$ for all strings x and y?
  - Is $(xy)z = x(yz)$ for all strings x, y and z?
- Denote by $rev(x)$ the reverse of a string x, e.g., $rev(abb) = bba$. Show using *mathematical induction*, that if $x = ay$, then $rev(x) = rev(y)\ a$.

  Start with $y = \varepsilon$ as the basis and proceed with the induction from there.
- Is $rev(xy) = rev(y)\ rev(x)$ for all strings x and y?

Strings and languages

jmsamaniego@uplb.edu.ph

- A (formal) language is a set of strings over some alphabet, e.g.,
  - $\mathbf{\Sigma}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$ over $\Sigma = \{0,1\}$
  - ODD = $\{x \in \mathbf{\Sigma}^*$: x has an odd number of a's and any number of b's$\}$ over $\mathbf{\Sigma} = \{a,b\}$
  - ID = $\{x \in \mathbf{\Sigma}^*$: x starts with a letter followed by 0 or more letters or digits$\}$ over $\mathbf{\Sigma} = \{a..z, 0..9\}$
  - INT = $\{x \in \mathbf{\Sigma}^*$: x is a sequence of one or more digits, prefixed by an optional + or - sign$\}$ over $\mathbf{\Sigma} = \{0..9, +, -\}$

Operations on languages

jmsamaniego@uplb.edu.ph

A language is a *set* of strings, so most set operations are applicable, along with other string-specific operations

- Union, $L_1 \cup L_2$ (also denoted as L_1+L_2)
- Intersection, $L_1 \cap L_2$
- Concatenation, $L_1L_2 = \{xy: x \in L_1 \text{ and } y \in L_1\}$
- Kleene Closure, $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
- Complement, $\Sigma^* - L$

Examples

- Concatenation of two finite languages

  $\{0,1\}\{0,1\} = \{00, 01, 10, 11\}$
- Kleene closure

  $\{1\}^* = \{\varepsilon, 1, 11, 111, 1111, \ldots\}$

  $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$

  $\{a, ab\}^* = \{\varepsilon, a, ab, aa, aab, aba, abab, \ldots\}$
- More concatenation examples

  $\{1\}^*\{0\} = \{0, 10, 110, 1110, 11110, \ldots\}$

  $\{1\}^*\{0\}\{1\}^* = \{0, 10, 01, 110, 101, 011, \ldots\}$

jmsamaniego@uplb.edu.ph

Operations on languages, examples

Recall: L = {a, ab, aba, abab, ababa, ...}

L = {a, aba, ababa, ...} ∪ {ab, abab, ababab, ...}
(a is a suffix) (b is a suffix)

= **a(ba)* + ab(ab)*** ← **regular expression**

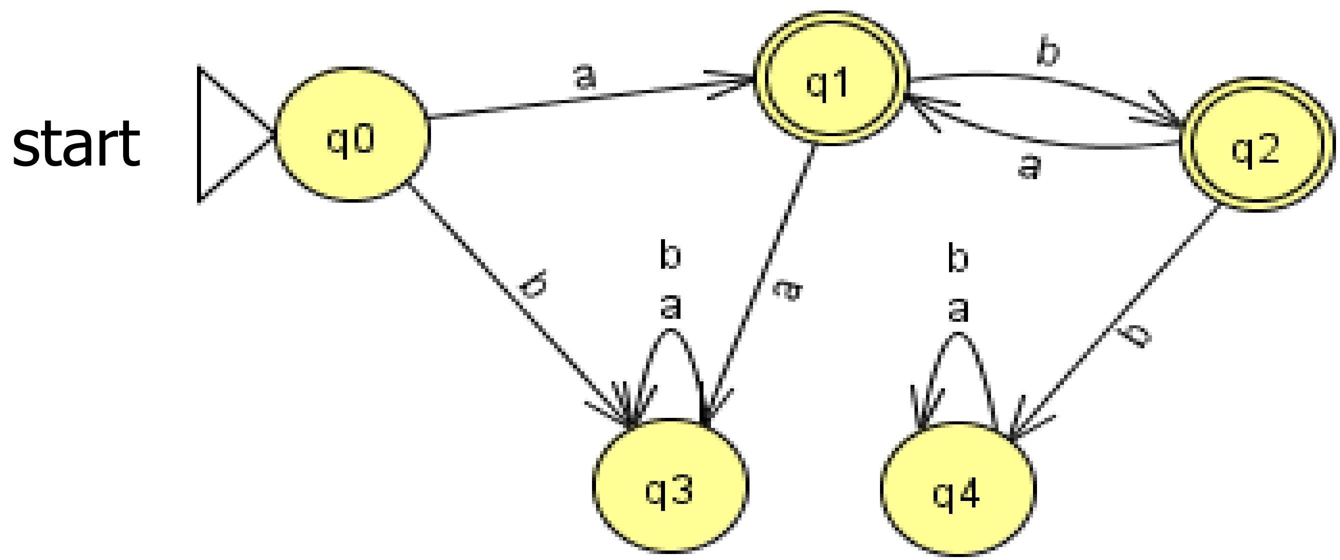
or

zero or more iterations

zero or more iterations

Languages = specifications, Automata = algorithm implementations

L = a(ba)* + ab(ab)*
= a ((ba)* + b(ab)*)



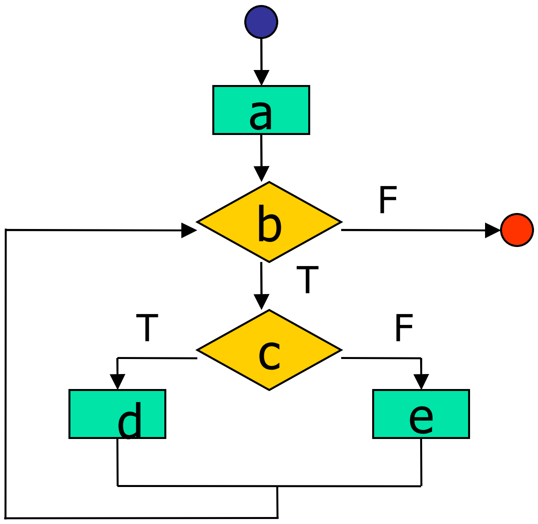
jmsamaniego@uplb.edu.ph

Sequence, selection and iteration in structured codes and regular expressions

jmsamaniego@uplb.edu.ph

- All procedural languages have basic control structures for sequence, selection and iteration

```
{ a;  // sample code
  while (b) {
      if (c) then d;
      else e;
  }
}
```

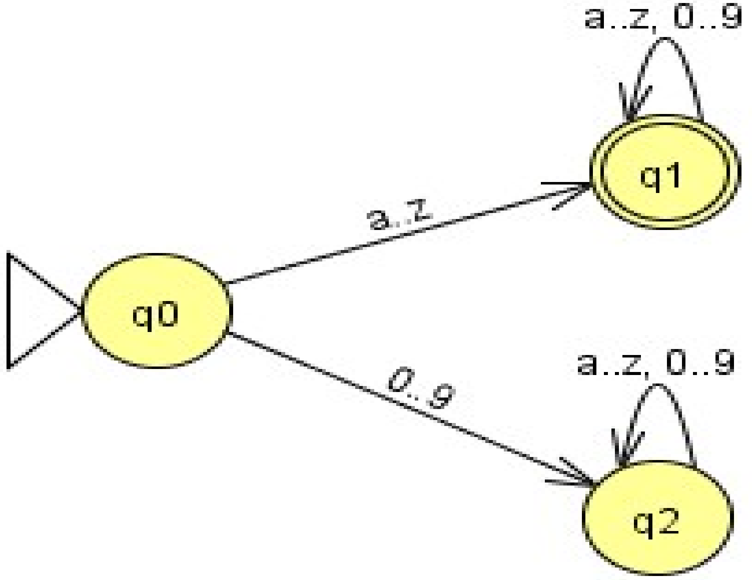


- regular expression that describes the flow of control in this code is: **a(bc(d+e))*b**

Another example

jmsamaniego@uplb.edu.ph

- ID = { $x \in \Sigma^*$: x starts with a letter followed by 0 or more letters or digits } over Σ = {a..z, 0..9}

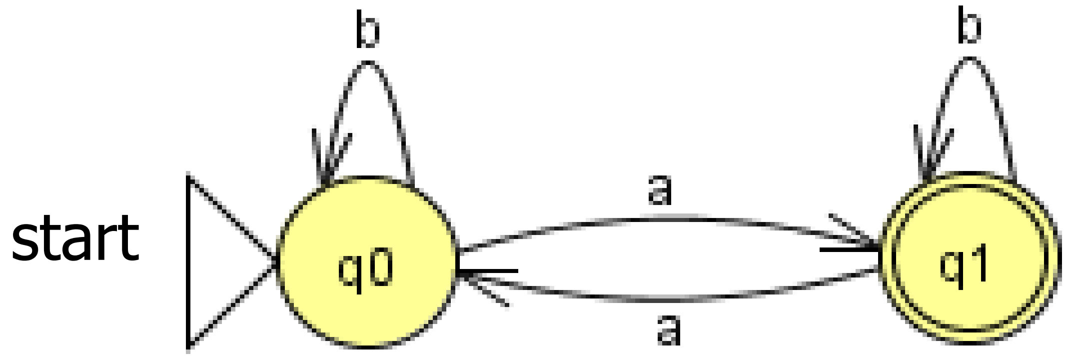


A valid regular expression is: **letter(letter+digit)***

Still another example

jmsamaniego@uplb.edu.ph

- ODD = { $x \in \Sigma^*$: x has an odd number of a's and any number of b's } over $\boldsymbol{\Sigma}$ = {a,b}



- A valid regular expression is: $\mathbf{b^*a(b^*ab^*a)^*b^*}$

Exercises

jmsamaniego@uplb.edu.ph

Draw deterministic finite automata that accept the ff. languages:

(a+b)*

(a+b)(a+b)*

a*+b*

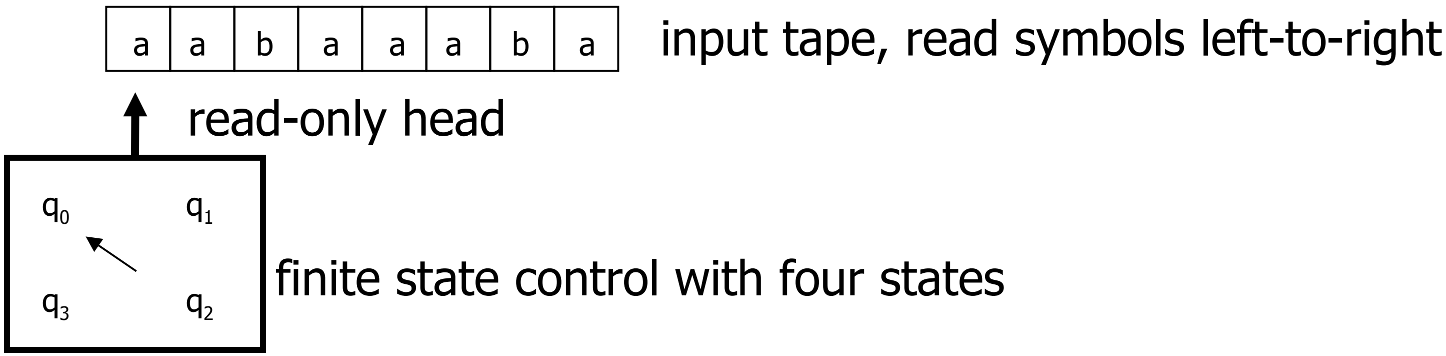
aa*+bb*

a*b*

a*(a+b)b*

Exercise: What relationships can you identify between these six languages?

DFA Deterministic Finite Automata



- Input string is read one symbol at a time
- Depending on the current state, and the current symbol scanned, the control may move into a new state; repeat until we reach the end of the string
- Input string is accepted if we end up in a final (or accepting) state

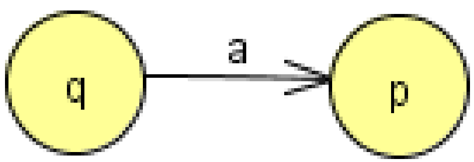
DFA, a formal definition

jmsamaniego@uplb.edu.ph

- a DFA is completely specified by the structure $M = (Q, \Sigma, \delta, q_0, F)$ where
  - Q is the finite set of states = $\{q_0, q_1, \ldots, q_n\}$
  - Σ is the input alphabet, e.g. $\{a, b\}$
  - δ is the transition function

    $\delta: Q \times \Sigma \rightarrow Q$

    (current State, symbol Scanned) → (new State)

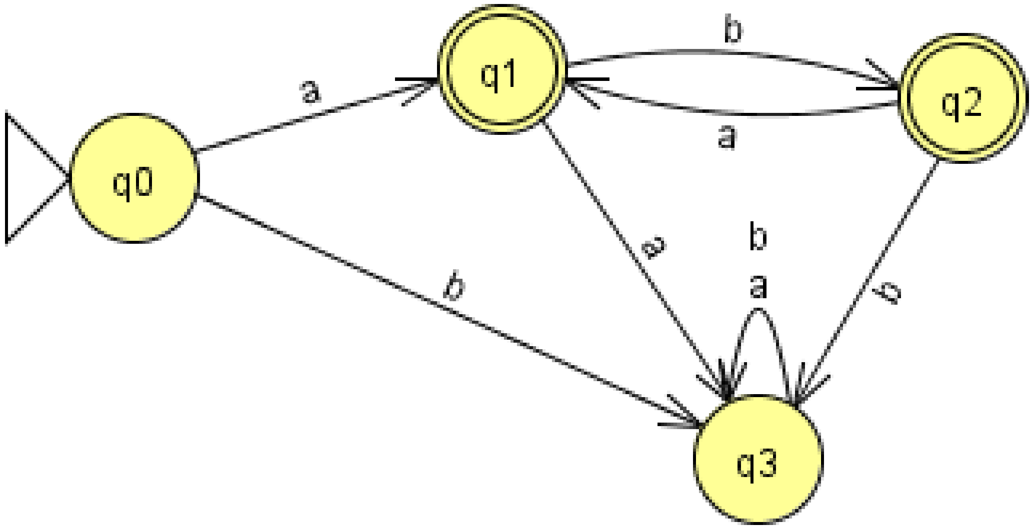


$\delta(q, a) = p$

  - q_0 is the start state, $q_0 \in Q$
  - F is the set of final states, $F \subseteq Q$

DFA, an example

jmsamaniego@uplb.edu.ph



$Q = \{q_0, q_1, q_2, q_3\}$

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

$F = \{q_1, q_2\}$

δ	$\downarrow q_0$	$\checkmark q_1$	$\checkmark q_2$	q_3
a	q_1	q_3	q_1	q_3
b	q_3	q_2	q_3	q_3

Instantaneous Descriptions of DFAs

- We represent the status of an execution of a DFA with the pair (*currentState, remainingInput*)
- Such a pair is known as an ID and provides a static snapshot of a dynamic process
- The acceptance of a string is demonstrated by a sequence of such IDs, e.g.

$$\left.\begin{array}{l}(q_0, aba) \vdash (q_1, ba) \\ \vdash (q_2, a)\end{array}\right\} \quad (q_0, aba) \vdash^* (q_1, \varepsilon\ )$$

"leads to"

"eventually leads to"

jmsamaniego@uplb.edu.ph

Acceptance by a DFA

$\vdash$

Let M = (Q, Σ, δ, q_0, F) be a DFA

- An input string x is *accepted* by M if there exists some final state p $\in$ F such that

$$(q_0, x) \vdash^* (p, \varepsilon)$$

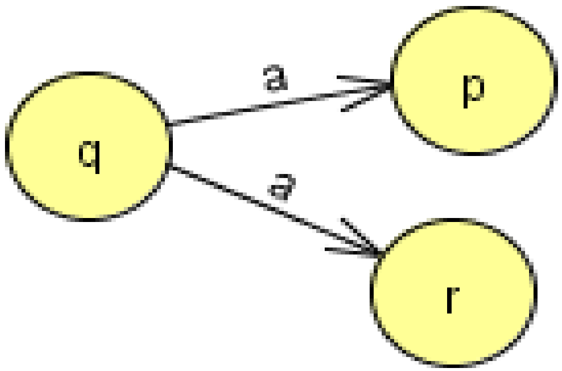
- The *language accepted* by M, denoted by L(M), is { x$\in\Sigma^*$: x is accepted by M }

jmsamaniego@uplb.edu.ph

NFAs
Non-deterministic Finite Automata

jmsamaniego@uplb.edu.ph

- So far we only allowed transitions that are *deterministic*, $\delta(q,a)$ always leads to a unique state
- Suppose we allow *non-deterministic* transitions of the form



- On input *a*, we can either go to state p or state r
- Note that we need to modify our definition of acceptance

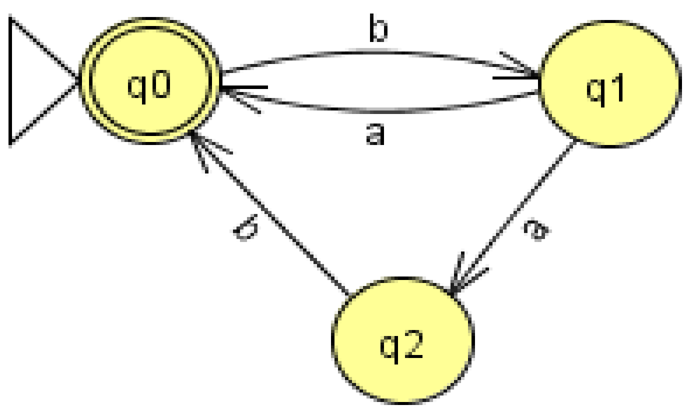
Why the need for non-determinism?

Use of non-determinism often simplifies the design of FA

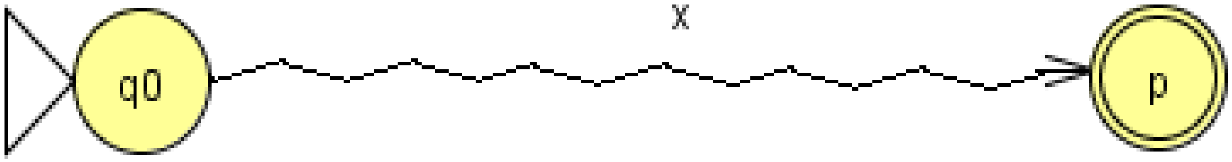
Ex. Consider the language **(ba + bab)***

- Simple NFA shown below
- *Challenge:* Find an equivalent DFA

start



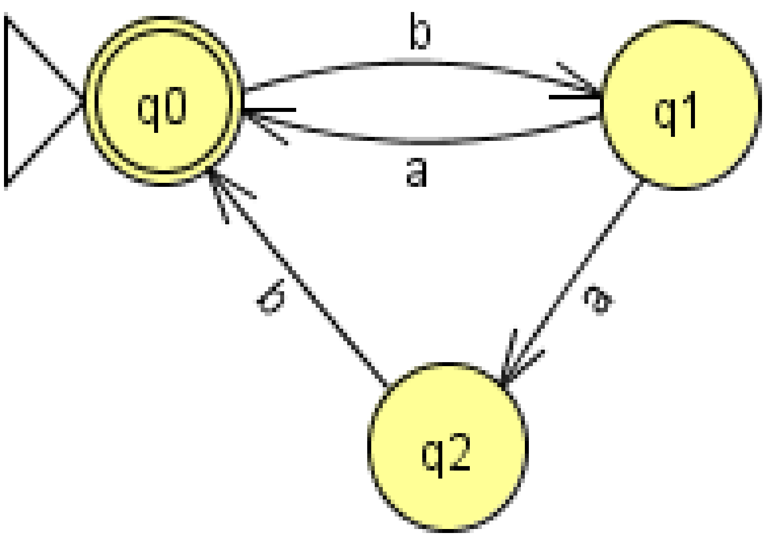
jmsamaniego@uplb.edu.ph



Acceptance in NFAs

- A string x is accepted by an NFA if *there is a path* that eventually ends in some final state
- Not all paths have to end up on a final state – just one path is enough to accept the string x

jmsamaniego@uplb.edu.ph



L = (ba + bab)**

In which nodes can we end up for the string bab?

$$(q_0, x) \vdash^* (p, \varepsilon), \quad p \in F$$

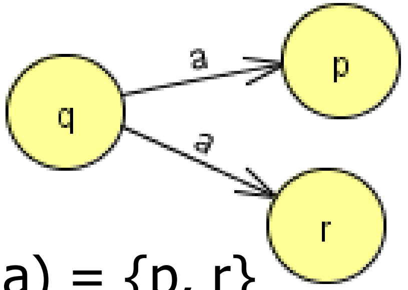
NFA, formal definition

jmsamaniego@uplb.edu.ph

- an NFA is a structure $M = (Q, \Sigma, \delta, q_0, F)$ where
  - Q is the finite set of states $= \{q_0, q_1, \ldots, q_n\}$
  - Σ is the input alphabet, e.g. $\{a, b\}$
  - δ is the transition function

$$\delta: Q \times \Sigma \rightarrow 2^Q$$

(current State, symbol Scanned) → (set of Possible New States)



$\delta(q, a) = \{p, r\}$

  - q_0 is the start state, $q_0 \in Q$
  - F is the set of final states, $F \subseteq Q$

DFA ⇔ NFA?

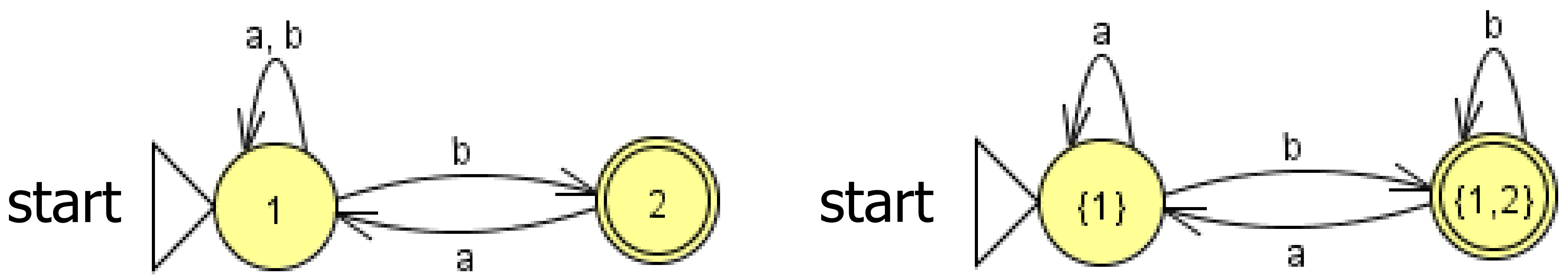
- Clearly, every DFA is an NFA in a trivial way (just don't allow choices)
- Questions: Can every NFA be converted to some equivalent DFA? If yes, how? Is there an algorithm to convert any NFA to an equivalent DFA?

jmsamaniego@uplb.edu.ph

Converting an NFA into a DFA

jmsamaniego@uplb.edu.ph

- We construct an equivalent DFA based on the possible states we can end up with on the NFA
- States in the DFA represent *sets of states* in the NFA



NFA → equivalent DFA

An NFA → DFA algorithm

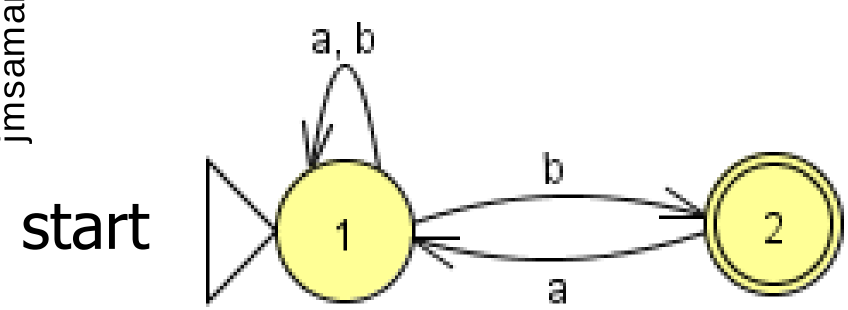
jmsamaniego@uplb.edu.ph

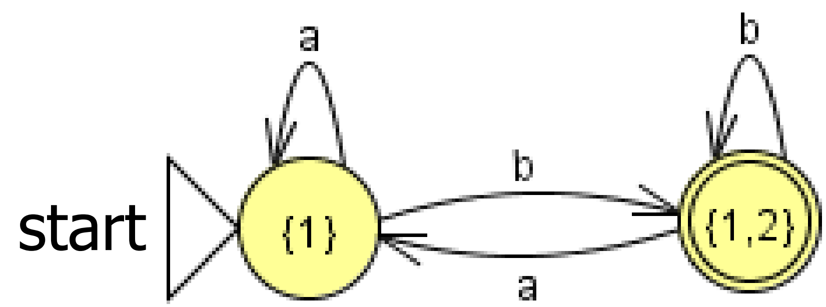
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be the given NFA
- We will construct an equivalent DFA

  $M' = (Q', \Sigma, \delta', q'_0, F')$
- Q' in the DFA will be a subset of 2^Q
- Starting with $q'_0 = \{q_0\}$, we build up M' state-by-state using the rules
  - $\delta'(q',a) = \bigcup_{q \in q'} \delta(q,a)$, for all $q' \in Q'$, $a \in \Sigma$
  - $q' \in F'$ if $q \in F$ and $q \in q'$

Too Greek?

- Starting with $q'_0 = \{q_0\}$, we build up M′ state-by- state using the rules
  - $\delta'(q',a) = \bigcup_{q \in q'} \delta(q,a)$, for all $q' \in Q'$, $a \in \Sigma$
  - $q' \in F'$ if $q \in F$ and $q \in q'$

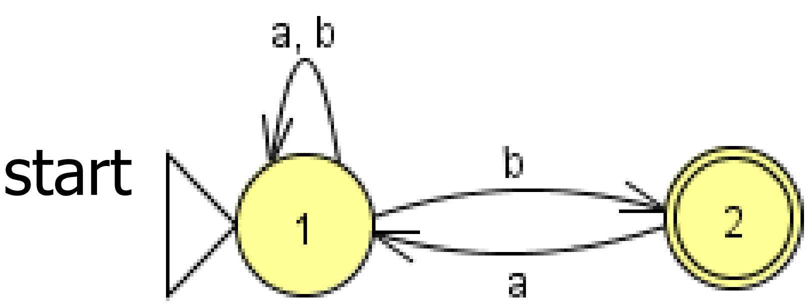


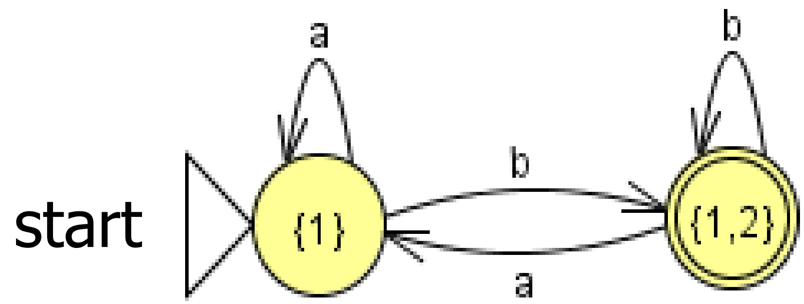


NFA → equivalent DFA

NFA → DFA conversion

jmsamaniego@uplb.edu.ph



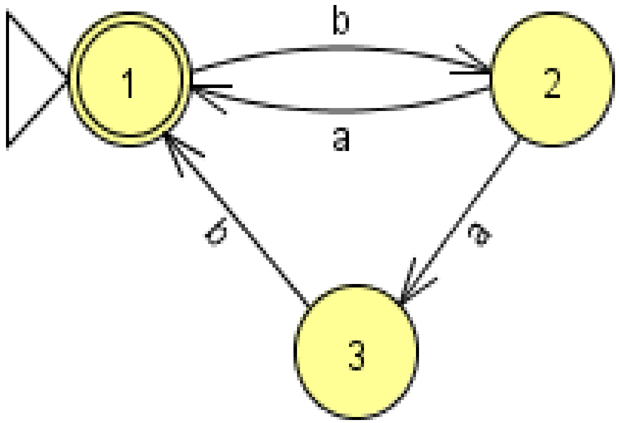


NFA → equivalent DFA

$\delta(1, a) = \{1\}$	$\delta'(\{1\}, a) = \{1\}$
$\delta(1, b) = \{1, 2\}$	$\delta'(\{1\}, b) = \{1, 2\}$
$\delta(2, a) = \{1\}$	$\delta'(\{1, 2\}, a) = \delta(1, a) \cup \delta(2, a) = \{1\}$
$\delta(2, b) = \varnothing$	$\delta'(\{1, 2\}, b) = \delta(1, b) \cup \delta(2, b) = \{1, 2\}$

Exercise: NFA → DFA

- Now try the algorithm for this NFA



δ	1	2	3
a	{}	{1,3}	{}
b	{2}	{}	{1}

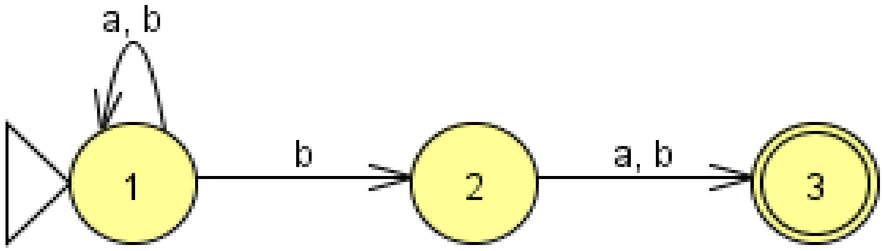
- Before you start, try to guess how many states will be in the equivalent DFA
- Be sure to check your work

jmsamaniego@uplb.edu.ph

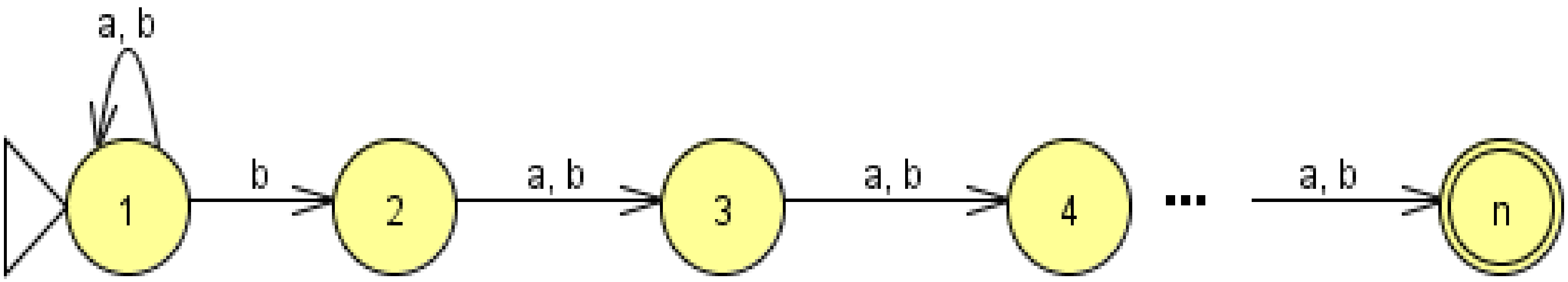
A bad case for NFA → DFA conversion

jmsamaniego@uplb.edu.ph

- The NFA below with n=3 states will require an exponential number of states in the DFA



- Can generalize this example for arbitrary n

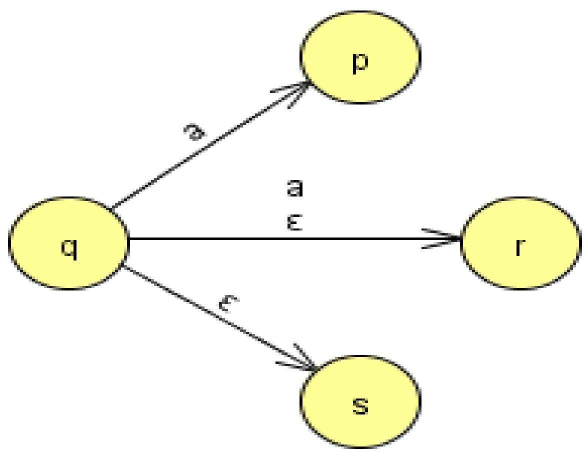


ε-NFAs
NFAs that allow moves on an empty string

- We further extend NFAs by allowing transitions on an empty string

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

(current State, symbol Scanned or Empty String)
→ (set of Possible New States)

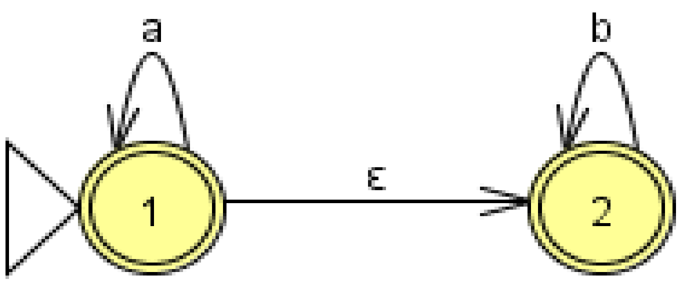


$\delta(q, a) = \{p, r\}$
$\delta(q, \varepsilon) = \{r, s\}$

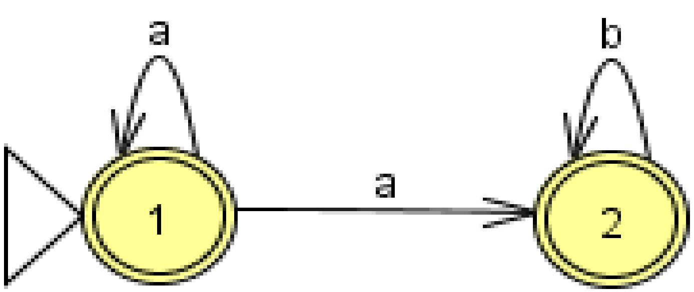
Like non-determinism, ε-moves often simplify the design of FA

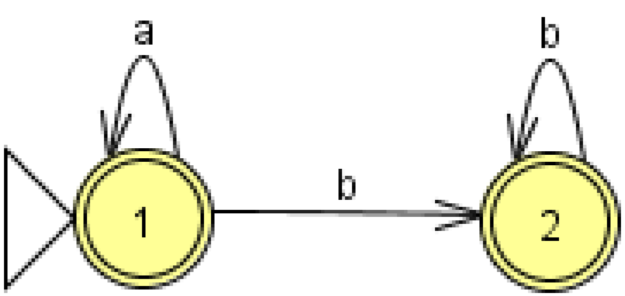
jmsamaniego@uplb.edu.ph

- Designing a FA for the language $\mathbf{a}^*\mathbf{b}^*$ is more intuitive if we allow ε-moves



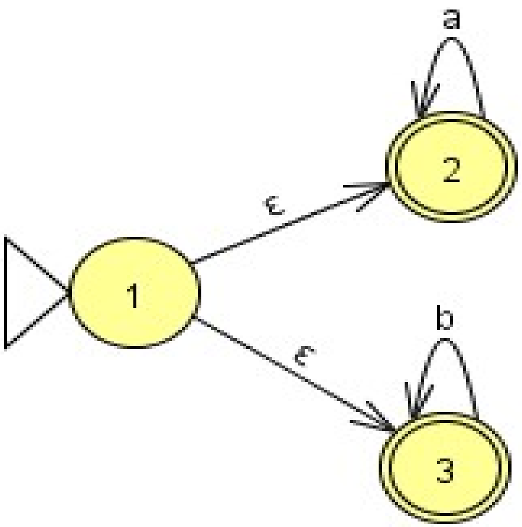
- Are the languages accepted by the NFAs below the same as $\mathbf{a}^*\mathbf{b}^*$? Why or why not?



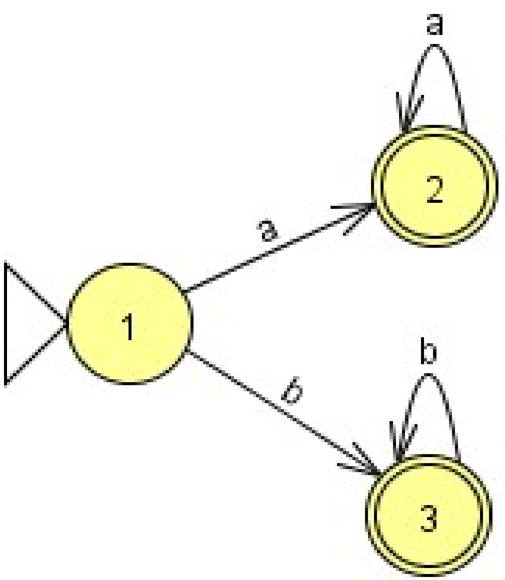


Another ε-NFA example

- **$a^* + b^*$**



What language does the related automaton on the right accept?



Exercise

jmsamaniego@uplb.edu.ph

- Give a regular expression and construct a finite automaton with ε-moves for accepting floating-point numbers over the alphabet { 0..9, +, -, ., e, E },

  e.g., 15., 1.5e1, 15e0, .15E2, 150e-1 $\in$ FLOAT
- Valid strings include
  - an optional sign
  - a string of digits (may be empty)
  - a decimal point
  - another string of digits (may be empty, but the digits before and after the decimal point *should not be both empty*)
  - an optional exponent (prefixed by 'E' or 'e' followed by an optional sign)

Exercises

jmsamaniego@uplb.edu.ph

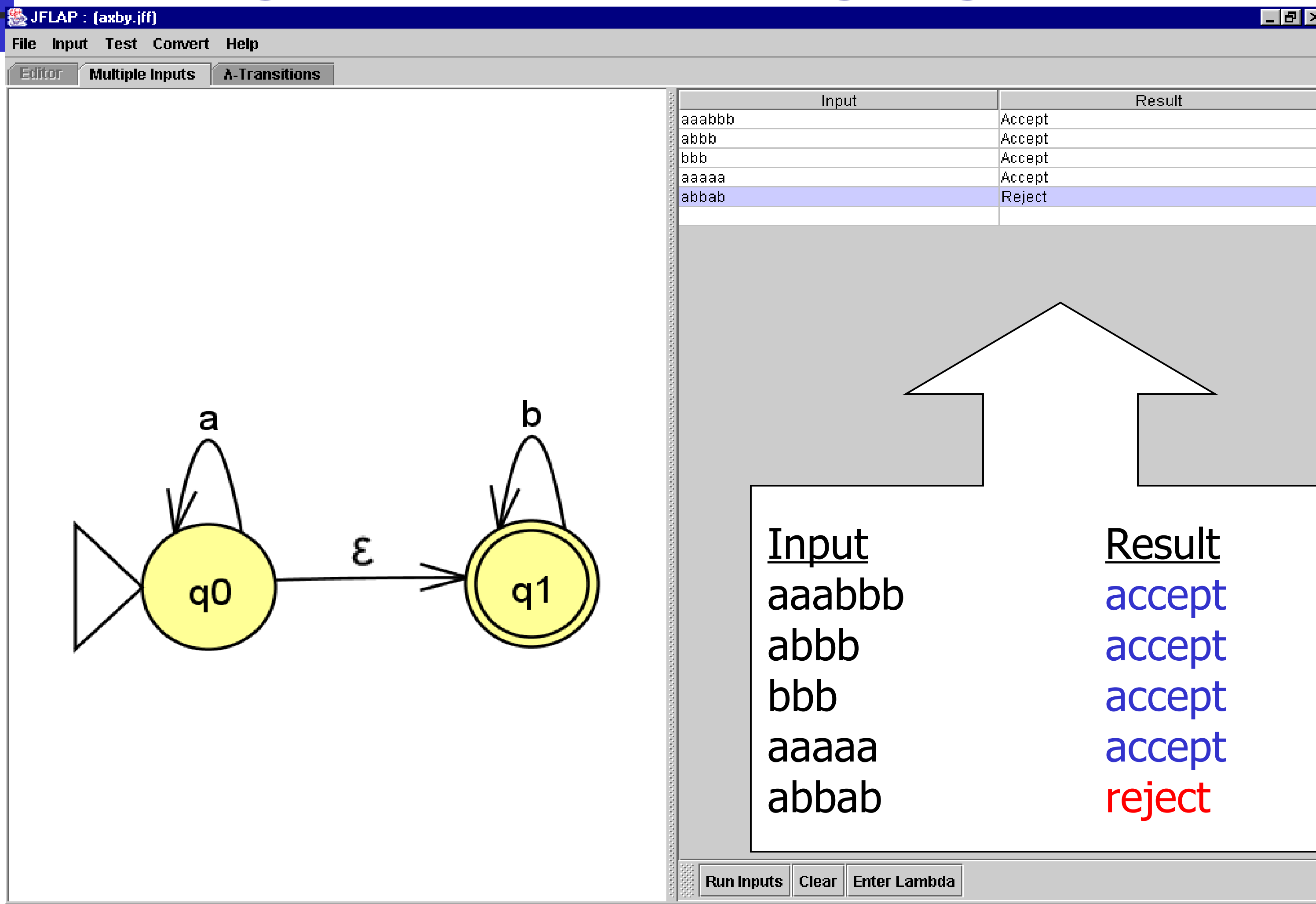
- Let Σ = {a, b}. Write regular expressions for all strings in Σ^*
  a) With no more than three a's
  b) with a number of a's that is divisible by 3,
  c) that has neither aa nor bb as a substring.

2. Construct equivalent DFAs that accept each of the three languages in #1.

- Use JFLAP to verify your solutions in #2.
- Construct a DFA for the language over Σ = {a, b} with an odd number of a's and an even number of b's. Then give an equivalent regular expression for the language accepted by your DFA.
- Give a DFA that accepts all strings over the binary alphabet Σ = {0, 1} which when interpreted as binary integers, are divisible by 3, (e.g., 001001 = 9 in decimal).

Using JFLAP
testing a FA for the language **a***b***

jmsamaniego@uplb.edu.ph



Input	Result
aaabbb	accept
abbb	accept
bbb	accept
aaaaa	accept
abbab	reject

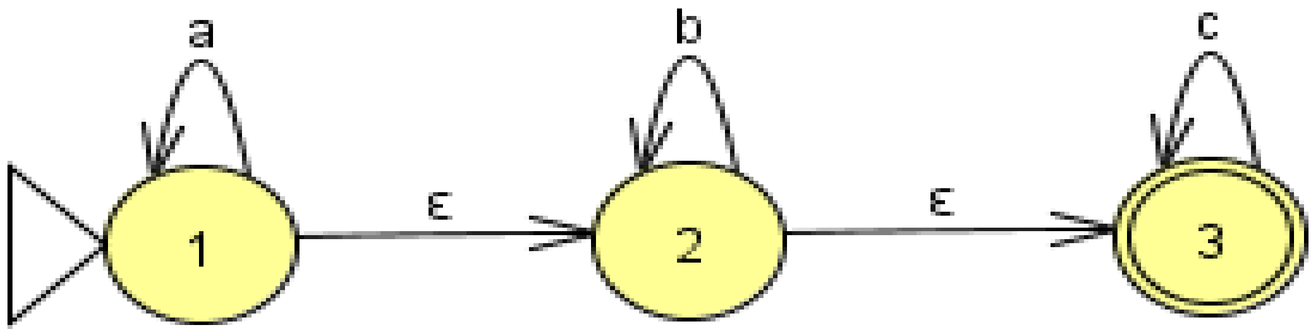
ε-closures and the elimination of ε-moves

- Informally, the ε-closure of a state *q* is the set of all states reachable from *q* via ε-moves
- A recursive definition of the ε-closure
  - $q \in \varepsilon\text{–close}(q)$
  - if $p \in \varepsilon\text{–close}(q)$ and $r \in \delta(p,\varepsilon)$ then $r \in \varepsilon\text{–close}(q)$
- Knowing the ε-closure of a state allows to find an equivalent NFA without ε-moves

a*b*c*

Example of ε-closures

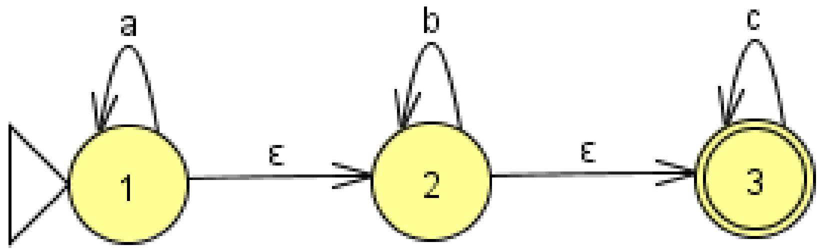
jmsamaniego@uplb.edu.ph



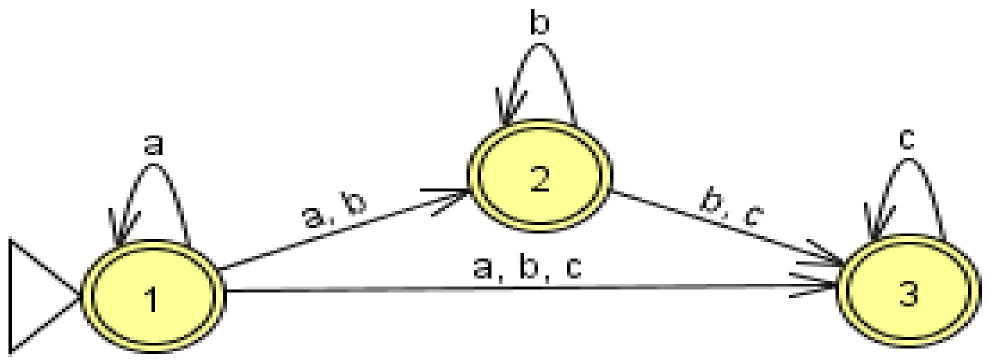
- ε–close(1) = {1, 2, 3}
- ε–close(2) = {2, 3}
- ε–close(3) = {3}

Converting ε-NFAs to NFAs

jmsamaniego@uplb.edu.ph



ε-NFA



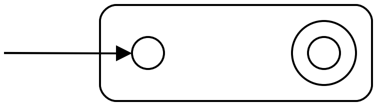
equivalent NFA
(without the ε-moves)

- $F' = F \cup \varepsilon\text{-close}(q_0)$, if $\varepsilon\text{-close}(q_0)$ contains some final state
- $\delta'(q,a)$ = all states reachable from q by paths labelled a or ε
- Exercise: also convert to DFA

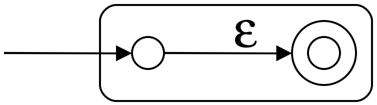
Regular expressions → ε-NFA

We show that any regular expression can be converted to an equivalent ε-NFA. We do this by **structural induction**:

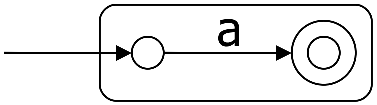
- **(Basis)** The ff. ε-NFAs are for the trivial languages
  ϕ, ε, and **a** (for any a ∈ Σ)



ϕ



ε

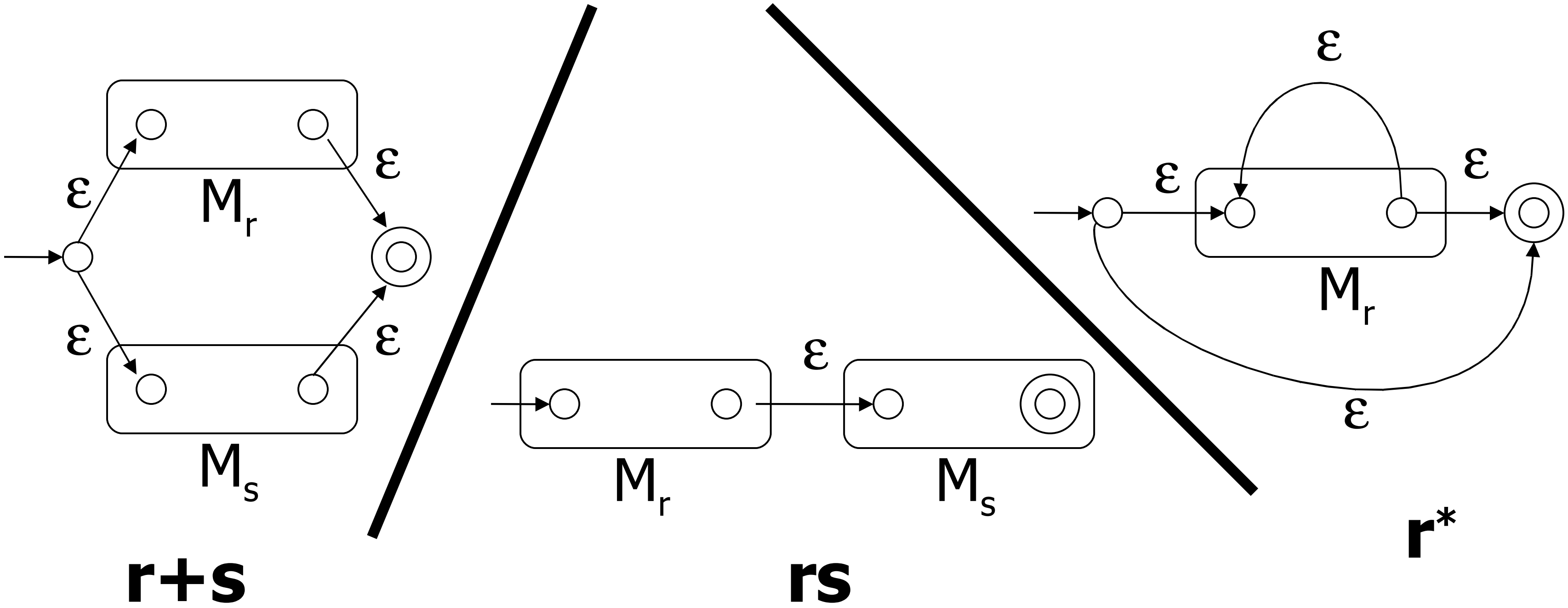


a

jmsamaniego@uplb.edu.ph

Regular expressions → ε-NFA

- **(Induction)** Let **r, s** be arbitrary regular expressions, with NFAs M_r and M_s. The ff. ε-NFAs are for **r+s**, **rs**, and **r***

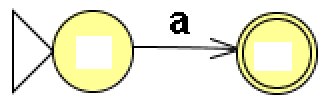


jmsamaniego@uplb.edu.ph

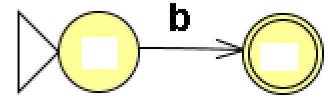
Example: Mechanical conversion of a regular expression to an ε-NFA

jmsamaniego@uplb.edu.ph

(ab+b)*ab

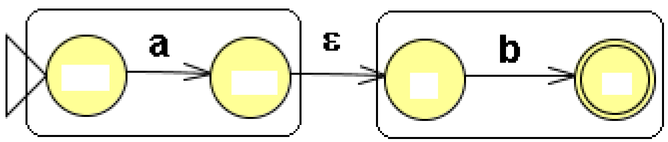


a

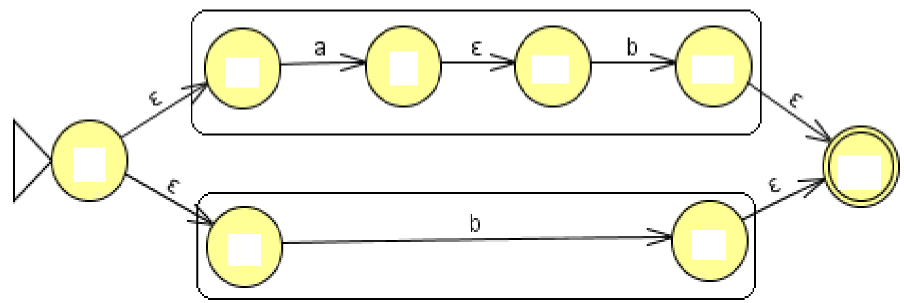


b

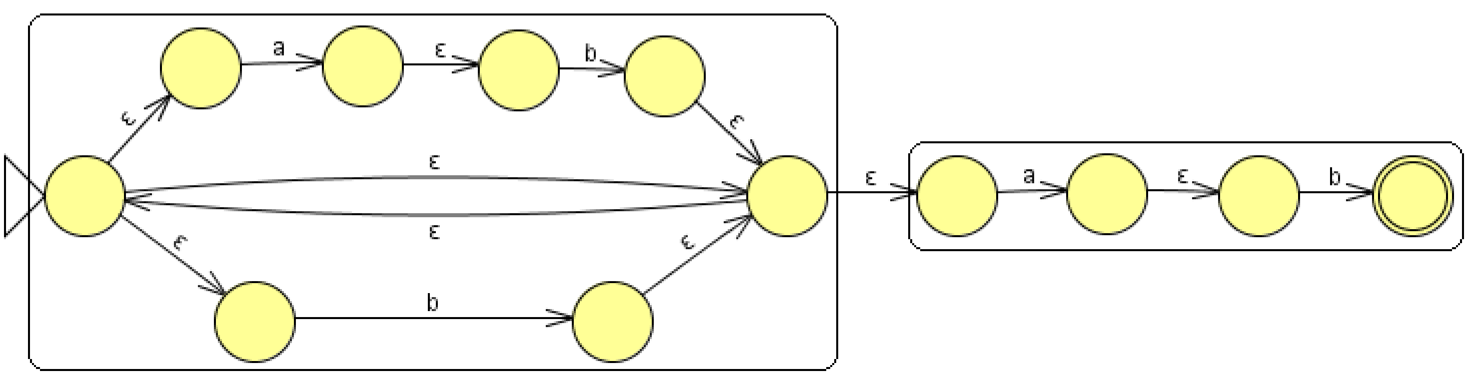
ab



ab+b



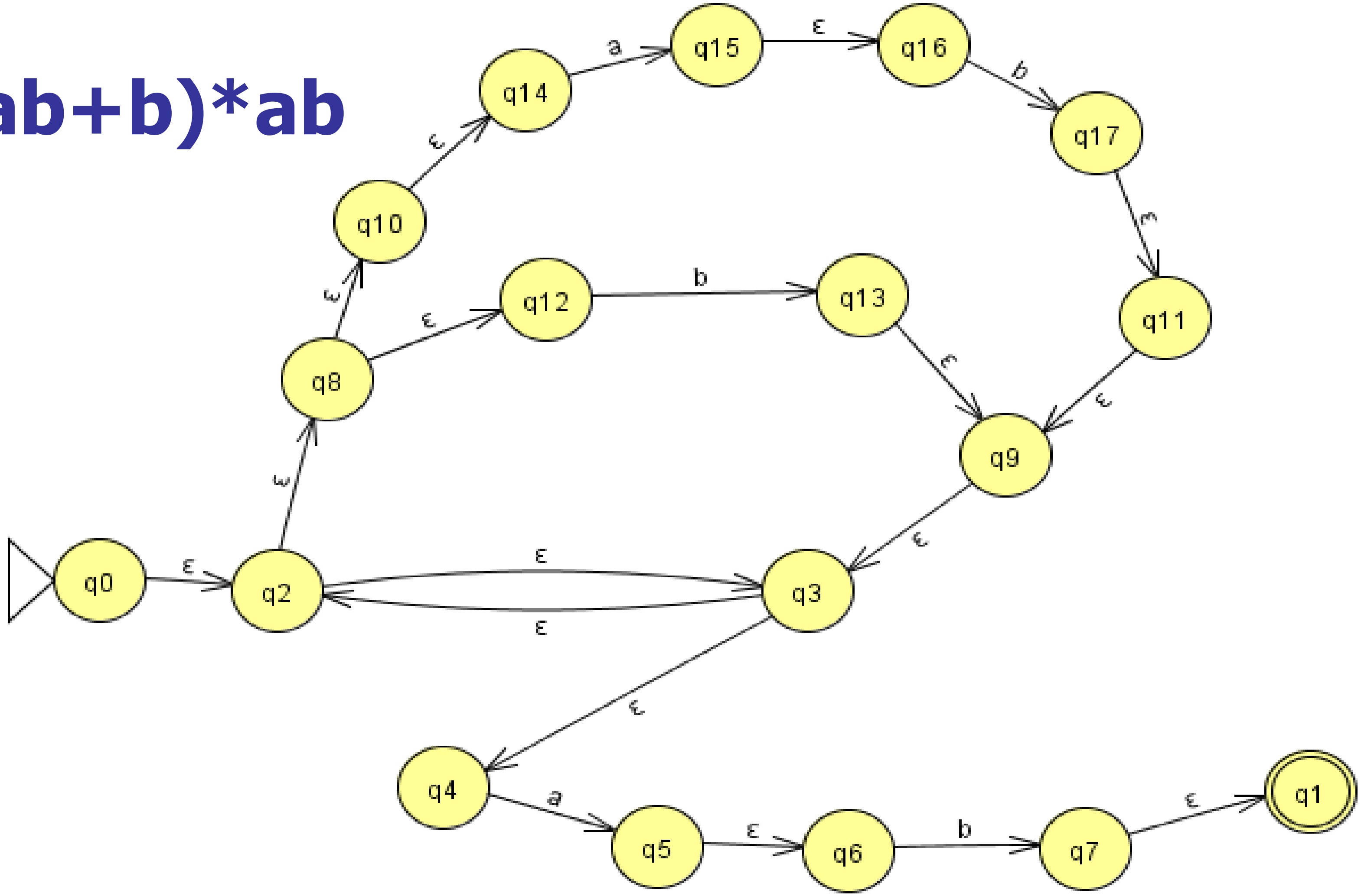
(ab+b)*ab



JFLAP actually produces an ε-NFA that is even bigger

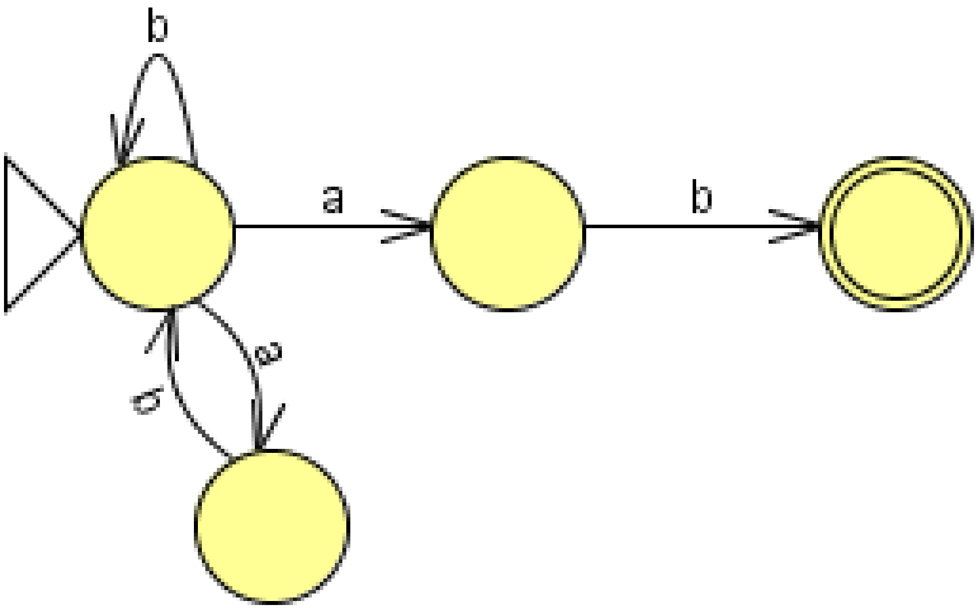
jmsamaniego@uplb.edu.ph

(ab+b)*ab



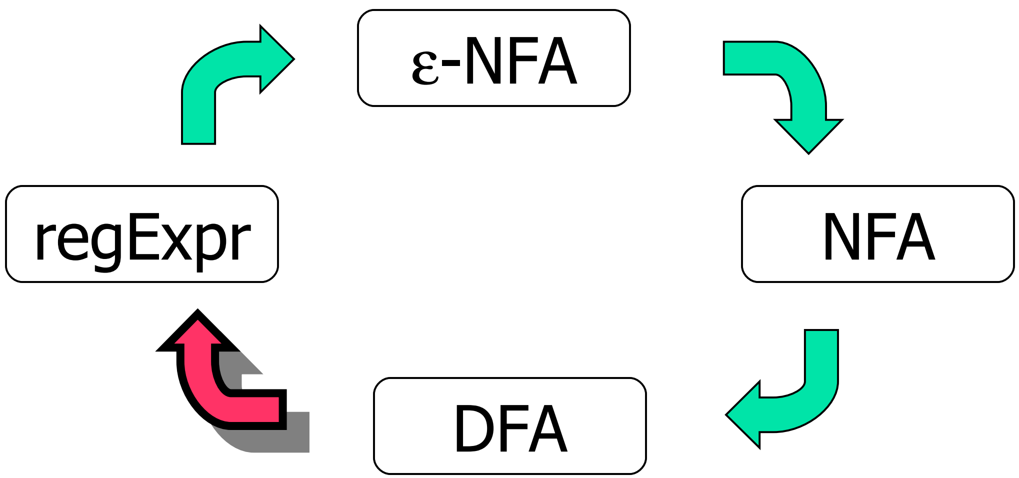
Handcrafted minimal NFA for (ab+b)*ab

jmsamaniego@uplb.edu.ph



Exercise: Find the minimal DFA equivalent to this NFA

Closing the circle:
DFA → regular expressions



- This would show that **DFAs, NFAs, ε-NFAs**, and **regular expressions** are all essentially equivalent

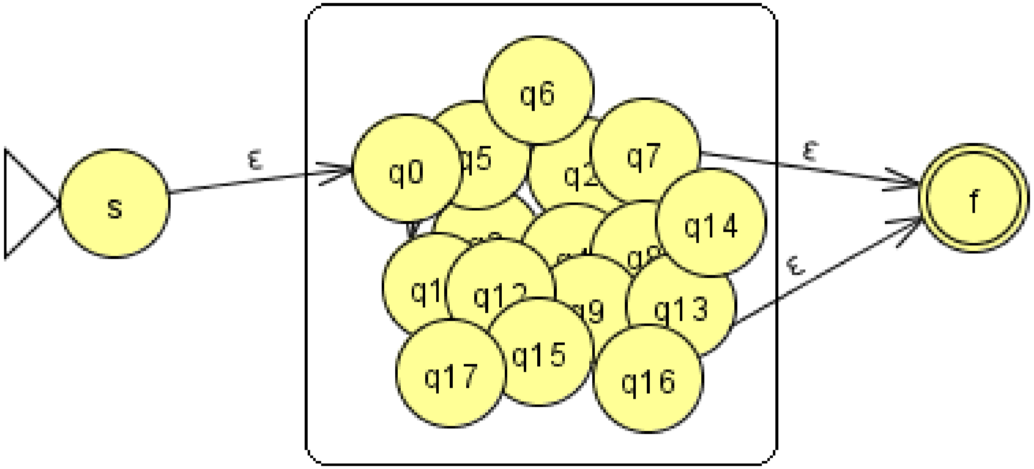
jmsamaniego@uplb.edu.ph

DFA → Regular Expression

using the state-elimination method

jmsamaniego@uplb.edu.ph

- First step is to add new "super-states" (s) and (f), with ε-moves from (s) to the original start state, and from the original final states to (f)



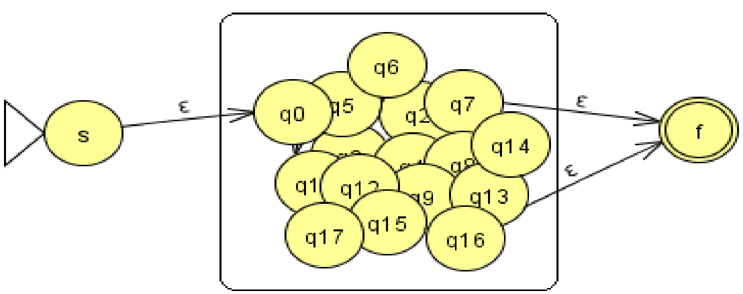
original DFA

DFA → Regular Expression

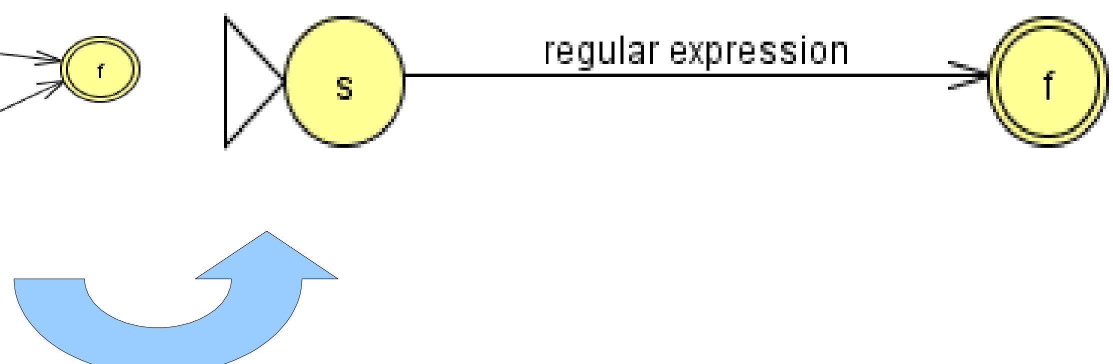
using the state-elimination method

jmsamaniego@uplb.edu.ph

- Then iteratively eliminate the original states, one state at a time, **replacing transition labels with corresponding regular expressions**, until only the super-states are left with a single transition labeled with the desired regular expression



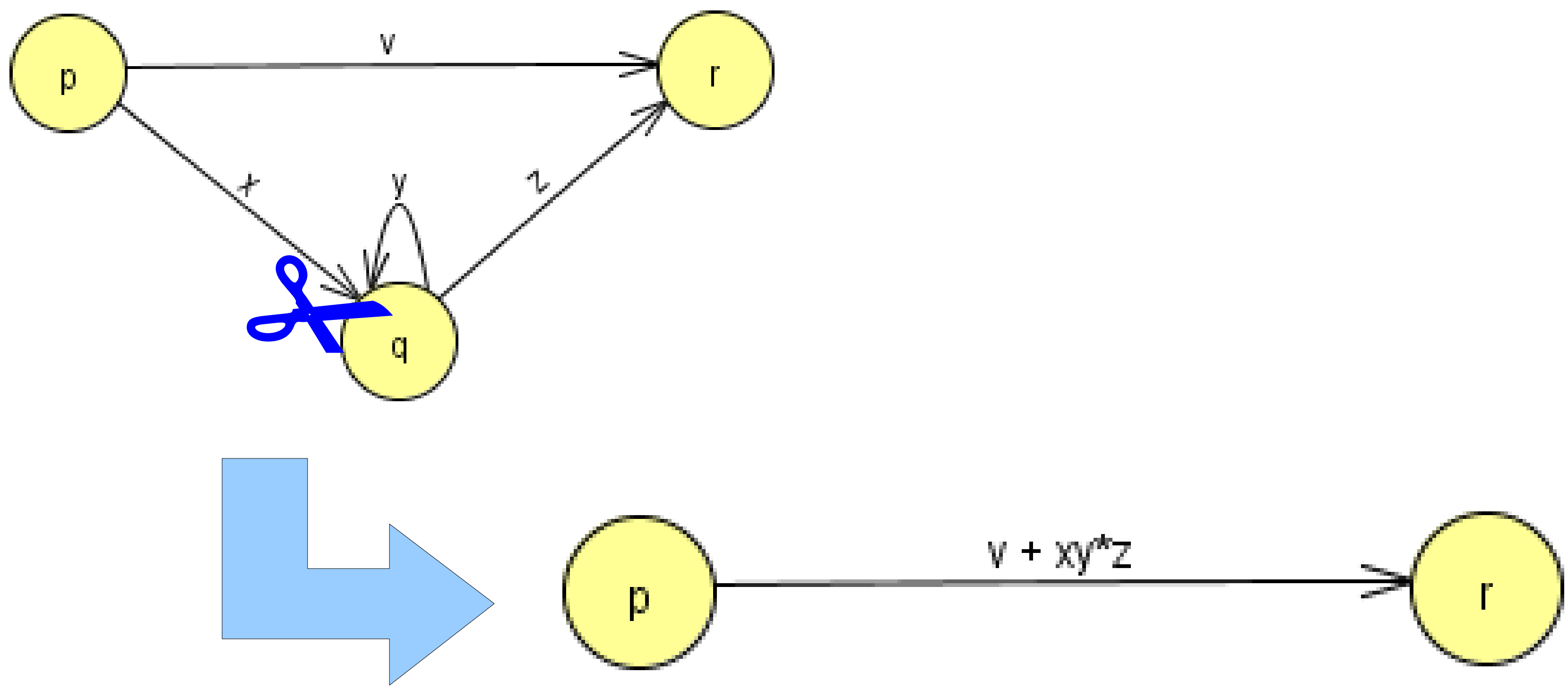
original DFA



DFA → Regular Expression
using the state-elimination method

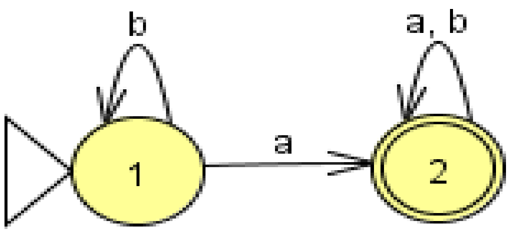
jmsamaniego@uplb.edu.ph

- Illustration on the **elimination of state q** below

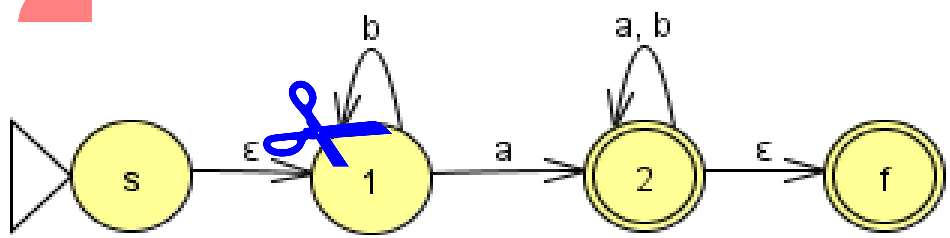


Easy DFA → RegEx example

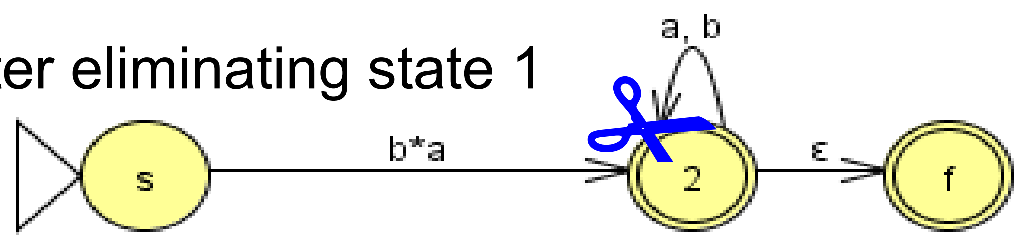
1 original DFA



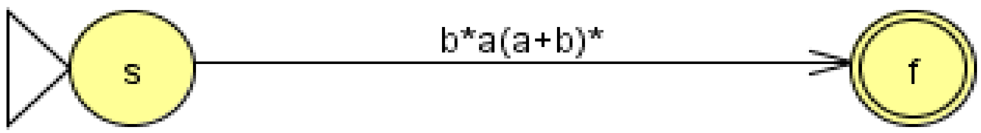
2 adding the super-states



3 after eliminating state 1



4 after eliminating state 2, we get the regular expression

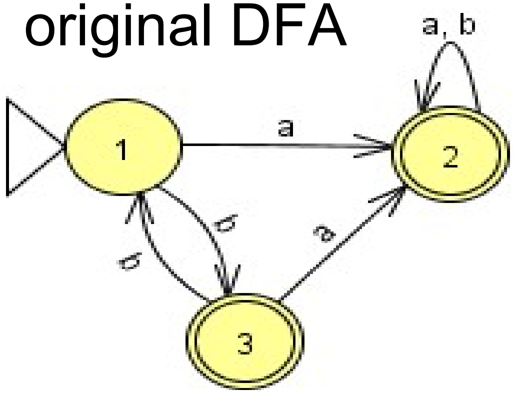


jmsamaniego@uplb.edu.ph

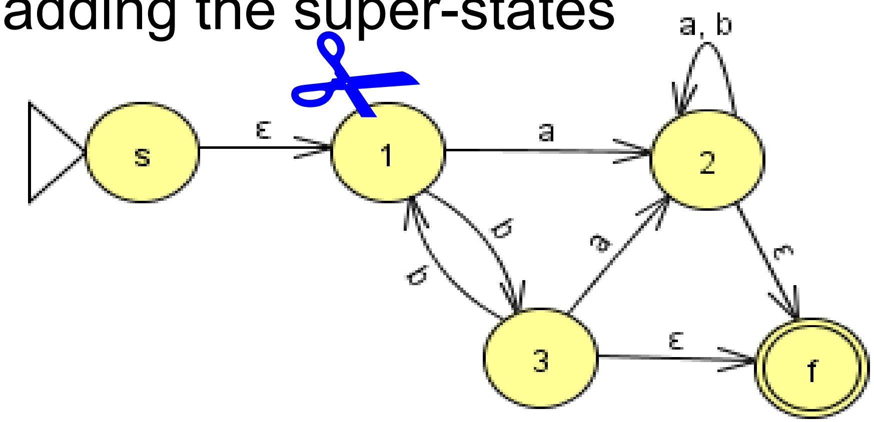
Another DFA → RegEx example

jmsamaniego@uplb.edu.ph

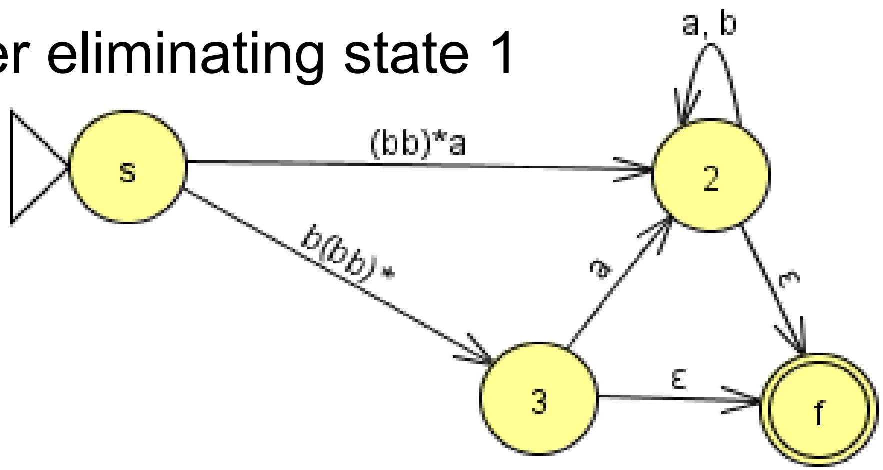
1 original DFA



2 adding the super-states



3 after eliminating state 1

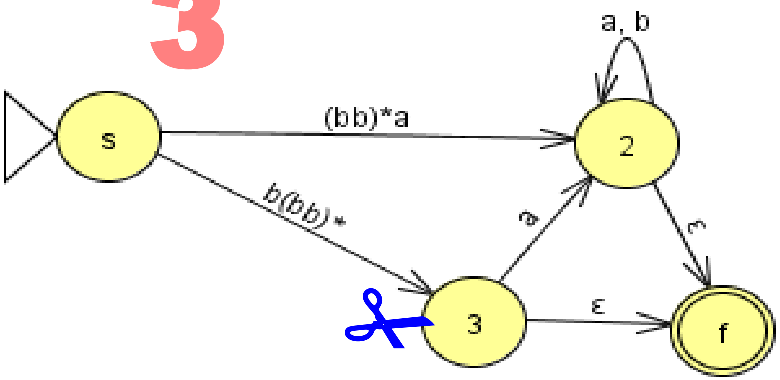


DFA → RegEx example

(continuation)

jmsamaniego@uplb.edu.ph

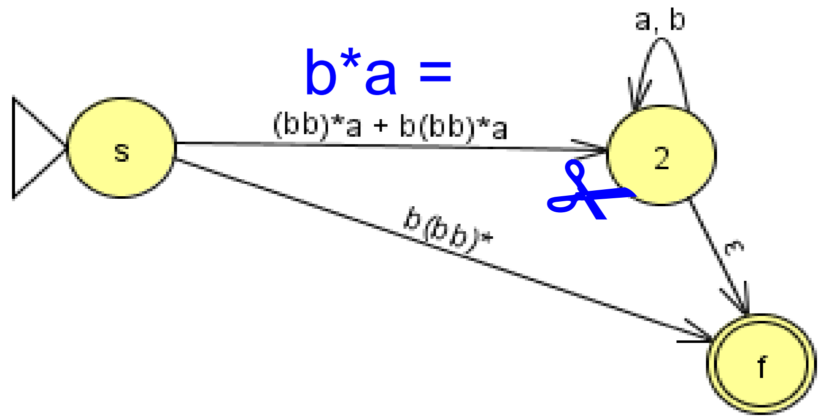
3



4

after eliminating state 3

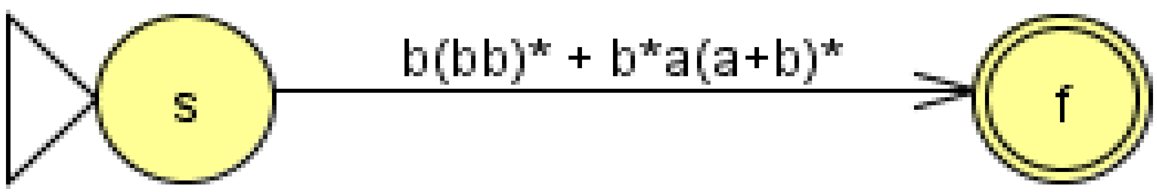
b*a =



after eliminating state 2,

we get the regular expression

5

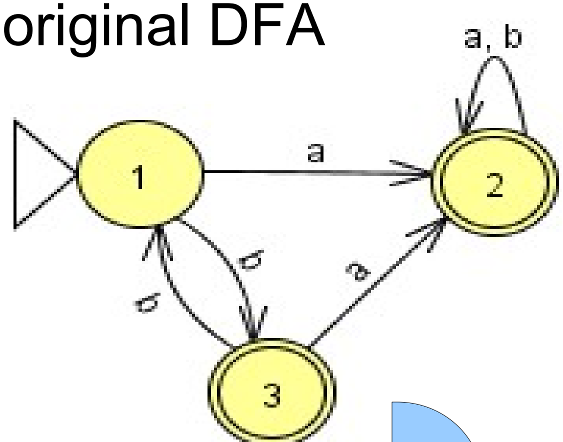


DFA → RegEx example

(continuation)

jmsamaniego@uplb.edu.ph

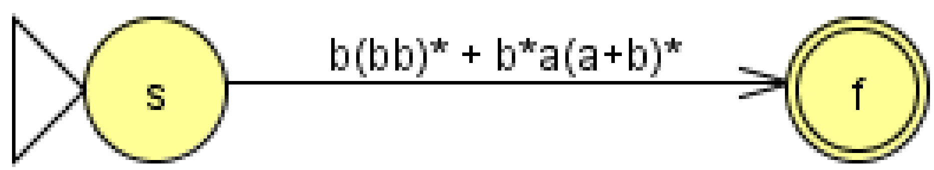
1 original DFA



Be sure to check your work...

you might find a simpler regular expression

5 resulting regular expression



Exercises

- Convince yourself that the order in which we eliminate the original states does not affect the resulting regular expression
- Use the state-elimination method to find an equivalent regular expression for the following DFA