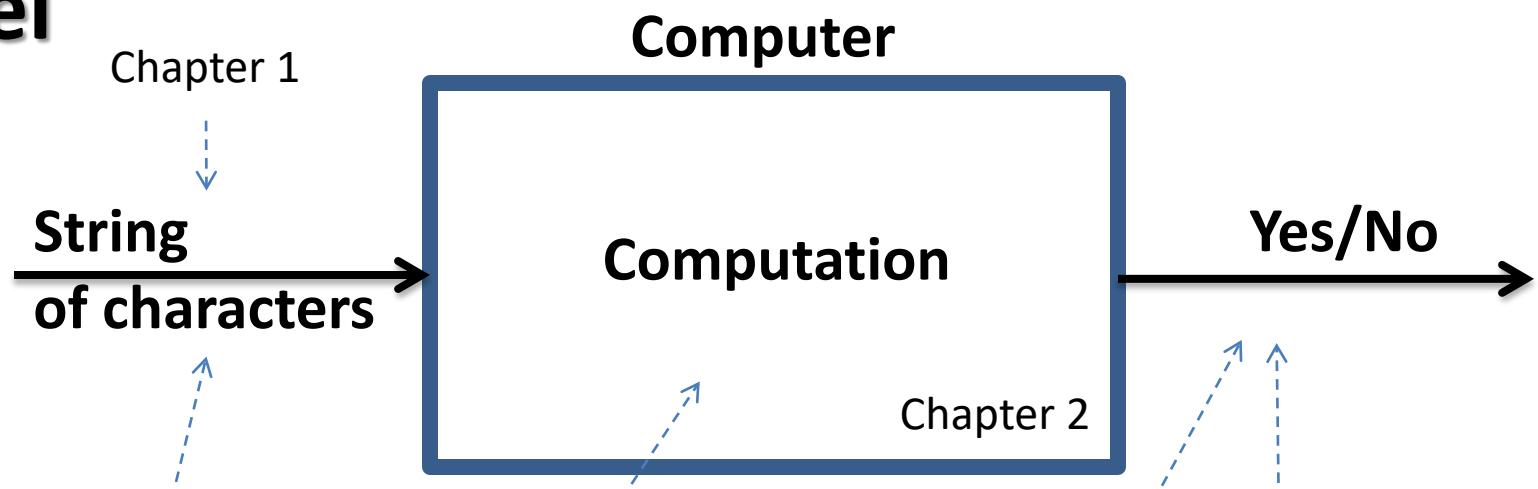
Chapter 2

Finite Automata and the Languages They Accept

You can start using JFLAP. It is helpful for understanding the concepts of computational models that we will learn in the next 2 months.
https://en.wikipedia.org/wiki/JFLAP
Note that you still need to use our notation in homework, quizzes and tests.

Model



This device plays a role of a language acceptor

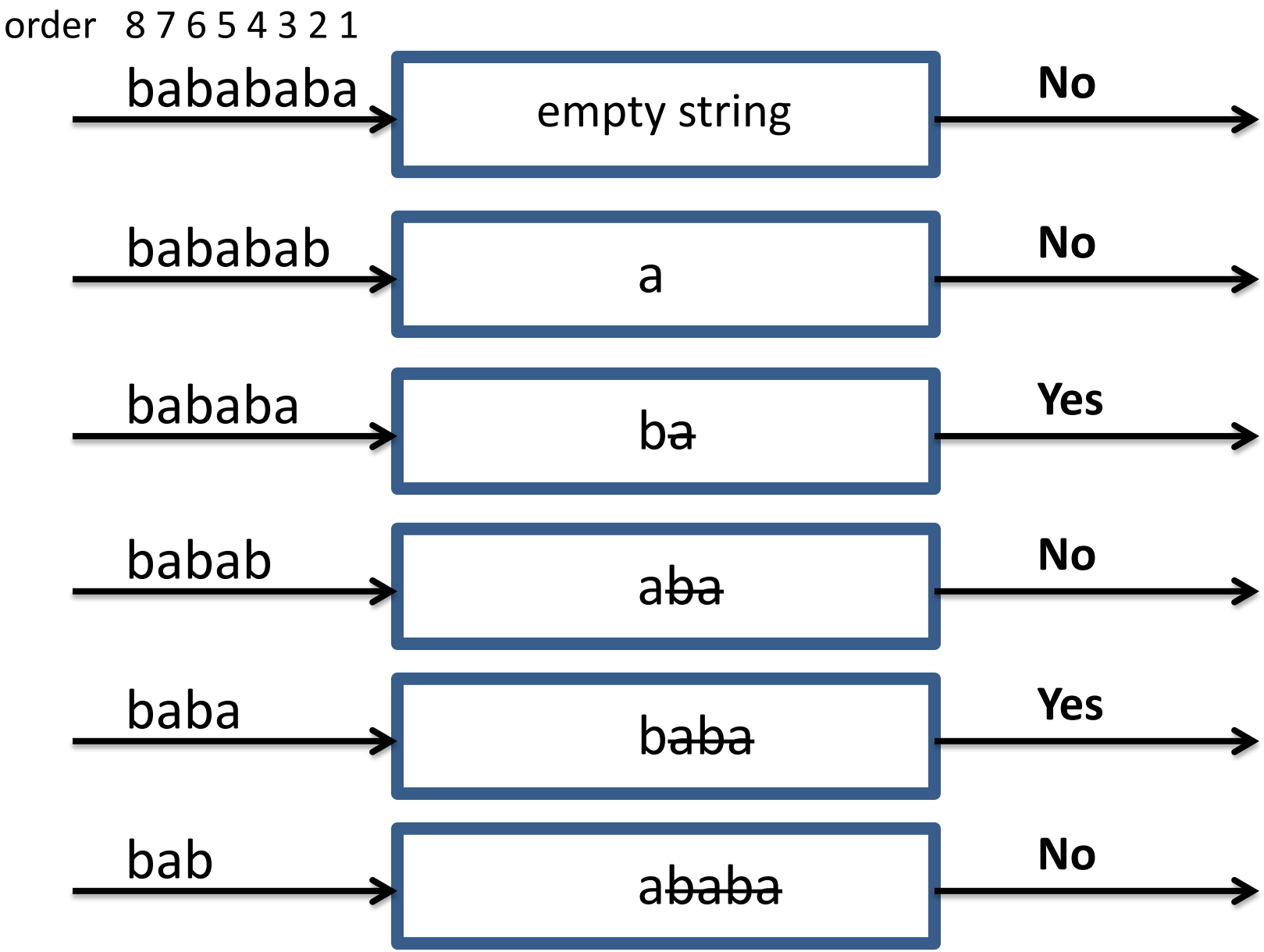
```
my $mystring;

$mystring = "Hello world!";

if($mystring =~ m/world/) { print "Yes"; } else { print “No”;}
```

Code in Perl

Intuition about finite automaton model requirements

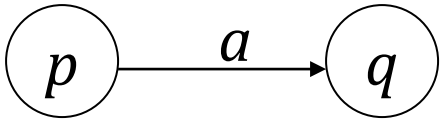
- A *finite automaton* is a simple type of computer
  - Its output is limited to “yes” or “no”
  - It has very primitive memory capabilities
- Our primitive computer that answers yes or no acts as a *language acceptor*
- For this model, consider that:
  - The input comes in the form of a string of individual input symbols
  - The computer gives an answer for the *current* prefix (the string of symbols that have been read so far)

Finite automaton that accepts language $L = \{(ab)^n \mid n \in N_{\geq 1}\}$.

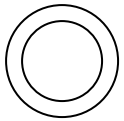
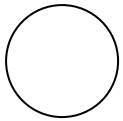


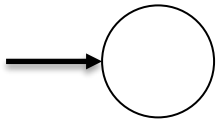
- A finite automaton (FA) or *finite state machine* is always in one of a finite number of *states*

- At each step FA makes a move (from state to state) that depends only on the **current state** and the **input symbol**



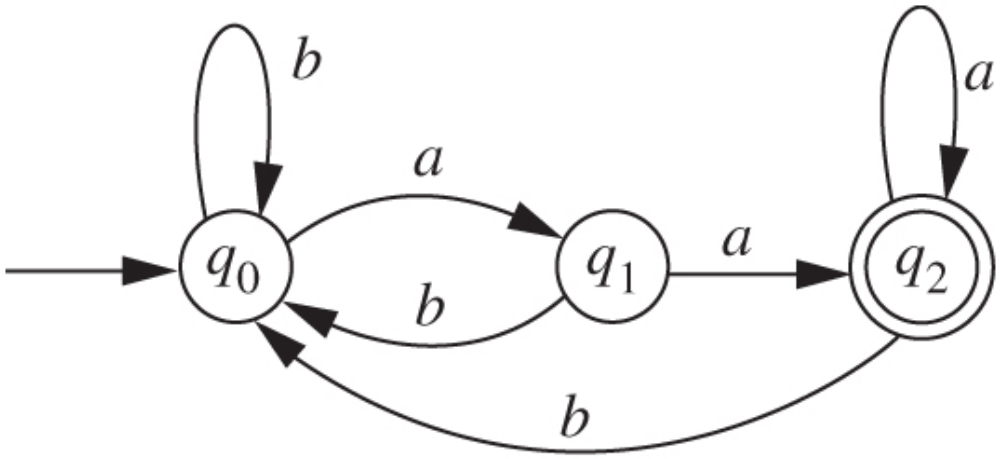
- The move is to enter a particular state (possibly the same as the one it was already in)

- States are either *accepting*  or *nonaccepting* 
  - Entering an accepting state means answering "yes"
  - Entering a nonaccepting state means "no"

- An FA has an initial state 

Finite Automata: Example

- This FA accepts the language of strings that end in *aa*
  - The three states represent strings that end with no *a*'s, one *a*, and two *a*'s, respectively
  - From each state, if the input is anything but an *a*, go back to the initial state, because now the current string doesn't end with *a*



Draw FA for language L={a,b}*

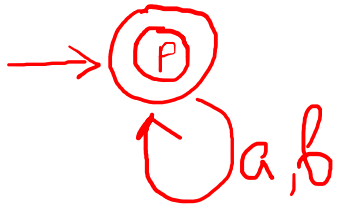
Draw FA for the empty language

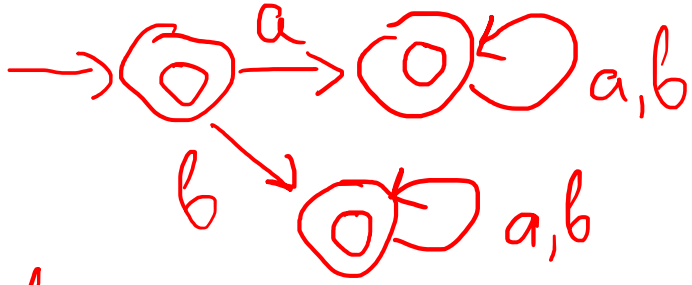
Draw FA for the language that contains only empty string and nothing else

Draw FA for the language L={b}* over alphabet {a,b}

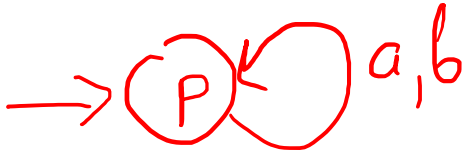
Draw FA for the language that contain strings ending with *b* and not containing *aa*

$\{a,b\}^*$

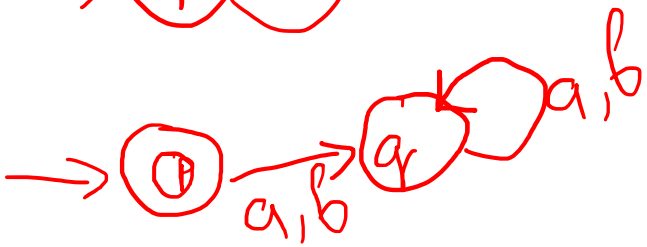




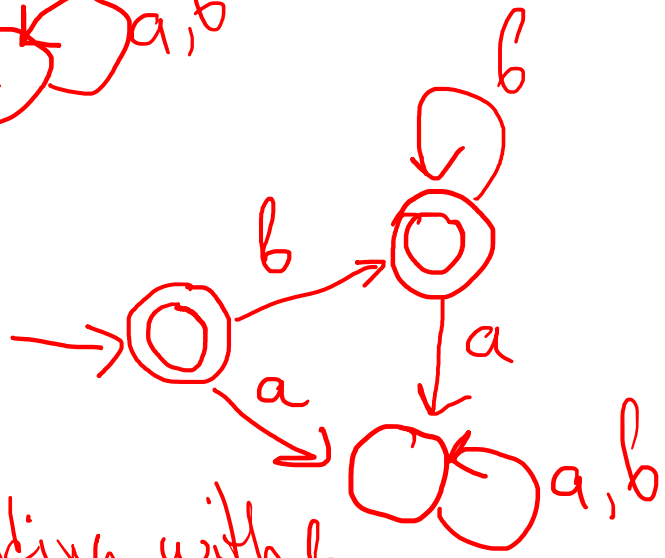
$\emptyset$

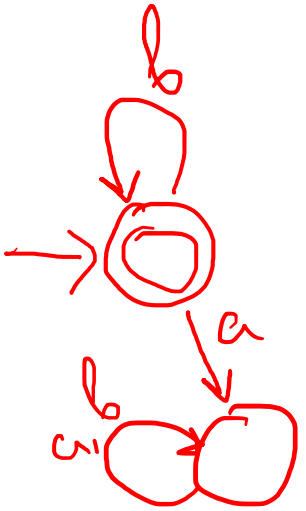


$\{\lambda\}$

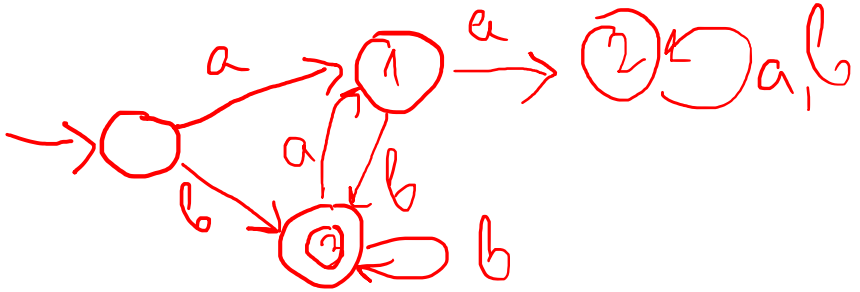


$\{b\}^*$ over $\{a,b\}$



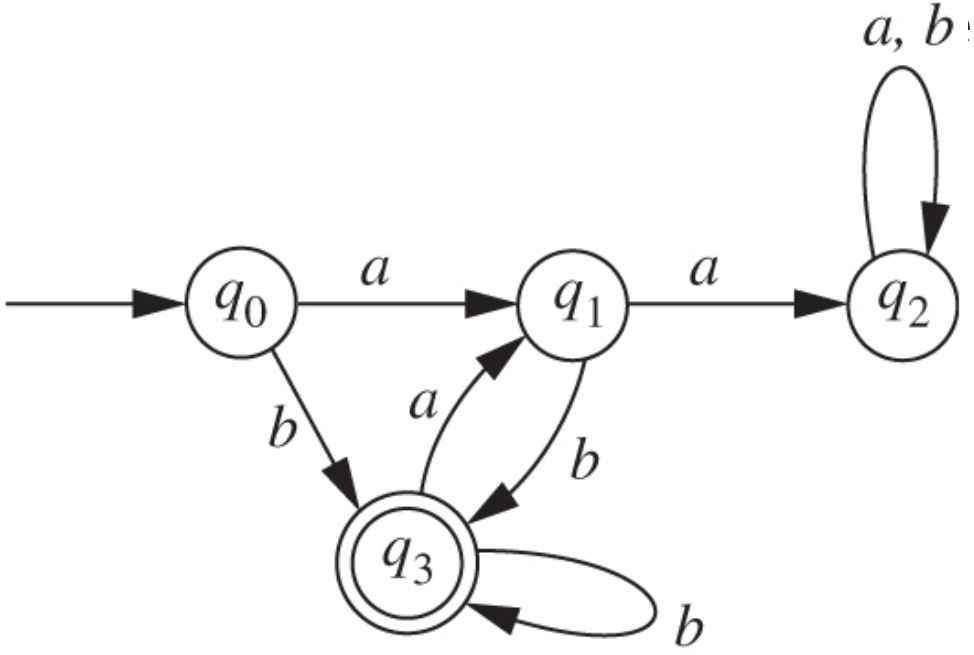


FA accepts strings ending with b and not containing aa



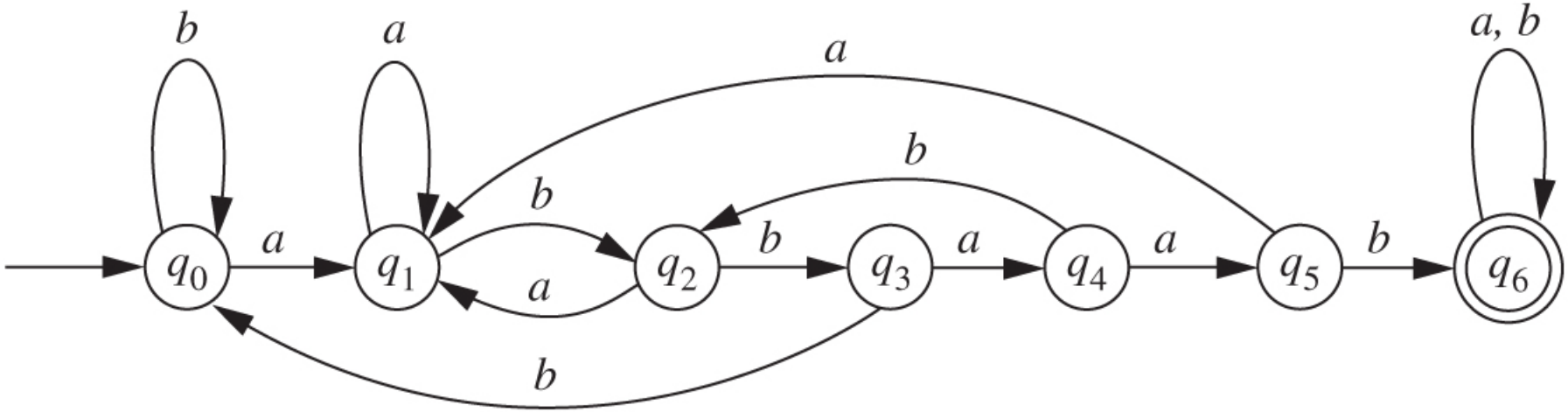
Finite Automata: Example

- This FA accepts the strings ending with *b* and not containing *aa*
  - The idea is to go to a permanently-non-accepting state if you ever read two *a*'s in a row
  - Go to an accepting state if you see a *b* (and haven't read two *a*'s),



Finite Automata: Example

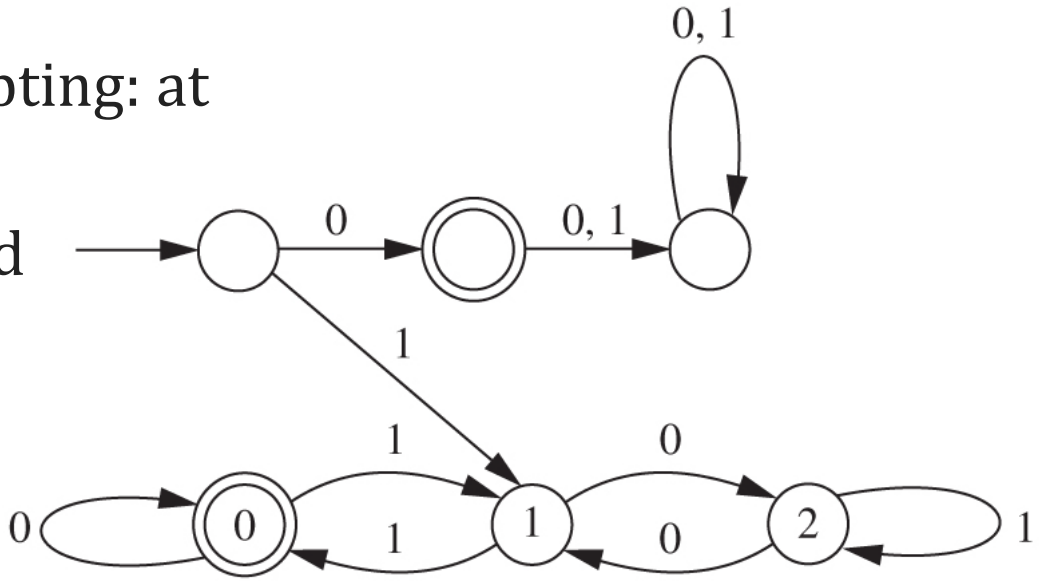
- This FA accepts strings that contain *abbaab*
- What do we do when a prefix of *abbaab* has been read but the next symbol doesn't match?
  - Go back to the state representing the longest prefix of *abbaab* at the end of the new current string
  - Example: If we've read *abba* and the next symbol is *b,* go to q_2, because *ab* is the longest prefix at the end of *abbab*



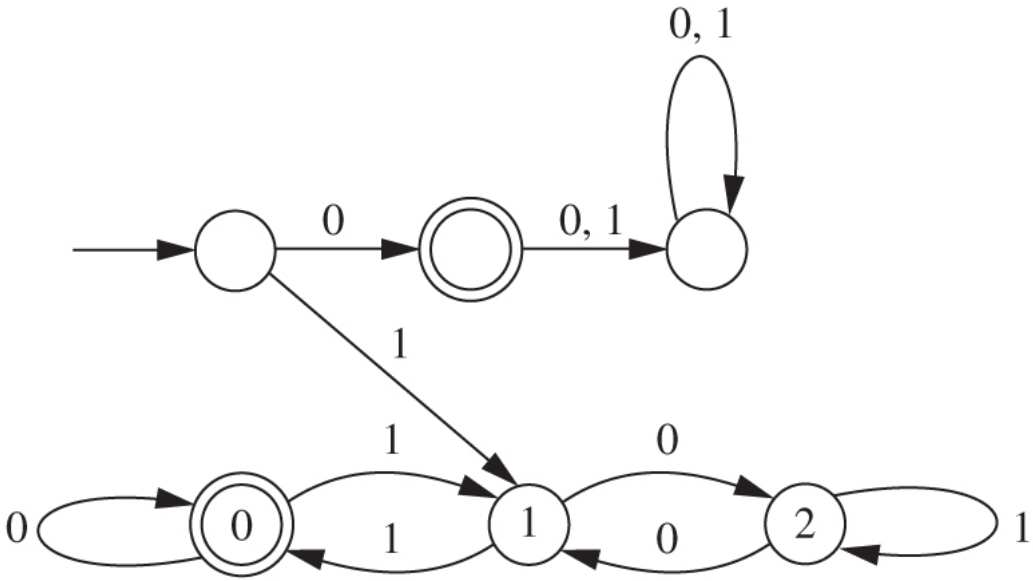
Finite Automata: the language of strings that are the binary representations of natural numbers divisible by 3.

If *x* represents *n*, and *n mod 3* is *r*, then what are *2n mod 3* and *(2n + 1) mod 3*? It is almost correct that the answers are *2r* and *2r + 1*; the only problem is that these numbers may be *3* or bigger, and in that case we must do another *mod 3* operation.

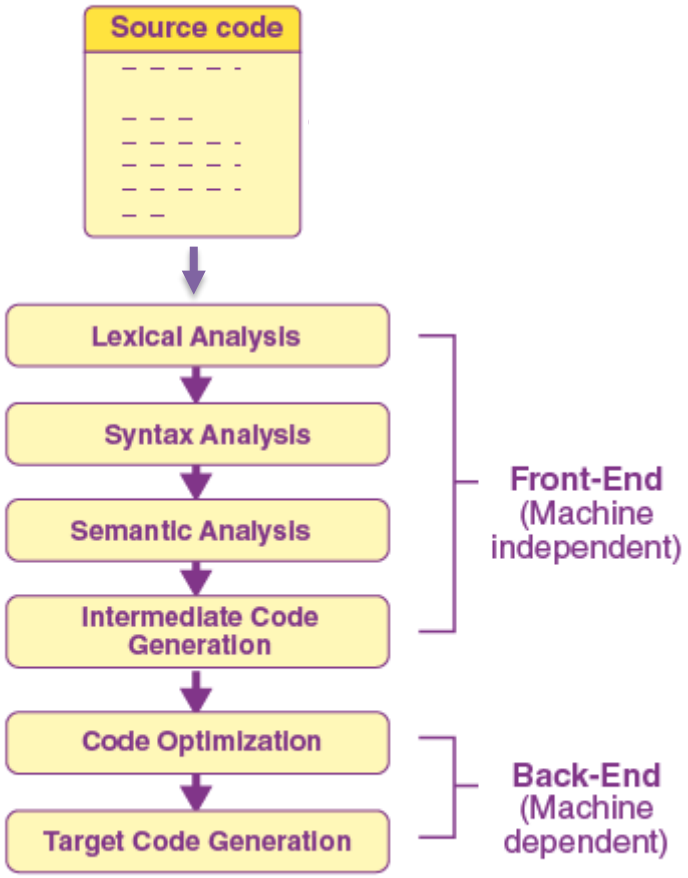
- States 0, 1, and 2 represent the current "remainder"
- The initial state is non-accepting: at least one bit is required
- Leading zeros are prohibited
- Transitions represent multiplication by two, then addition of the input bit

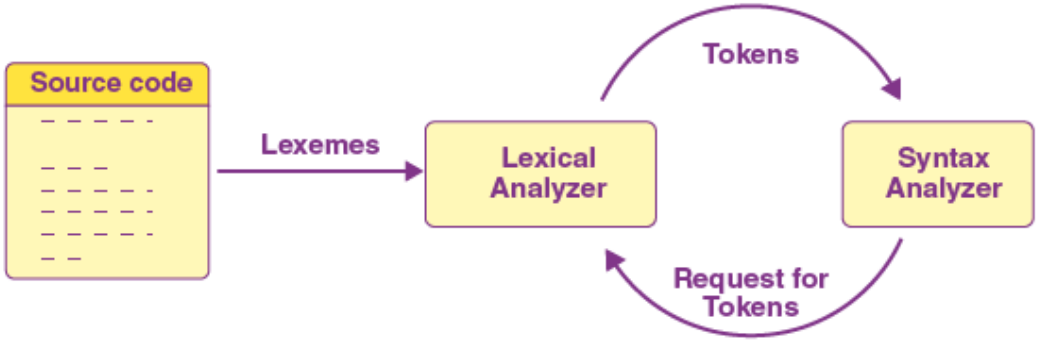


n	bin	r	n	bin	r
0	0	0	16	10000	1
1	1	1	17	10001	2
2	10	2	18	10010	0
3	11	0	19	10011	1
4	100	1	20	10100	2
5	101	2	21	10101	0
6	110	0	22	10110	1
7	111	1	23	10111	2
8	1000	2	24	11000	0
9	1001	0	25	11001	1
10	1010	1	26	11010	2
11	1011	2	27	11011	0
12	1100	0	28	11100	1
13	1101	1	29	11101	2
14	1110	2	30	11110	0
15	1111	0	31	11111	1



Finite Automata: Lexical Analysis Example



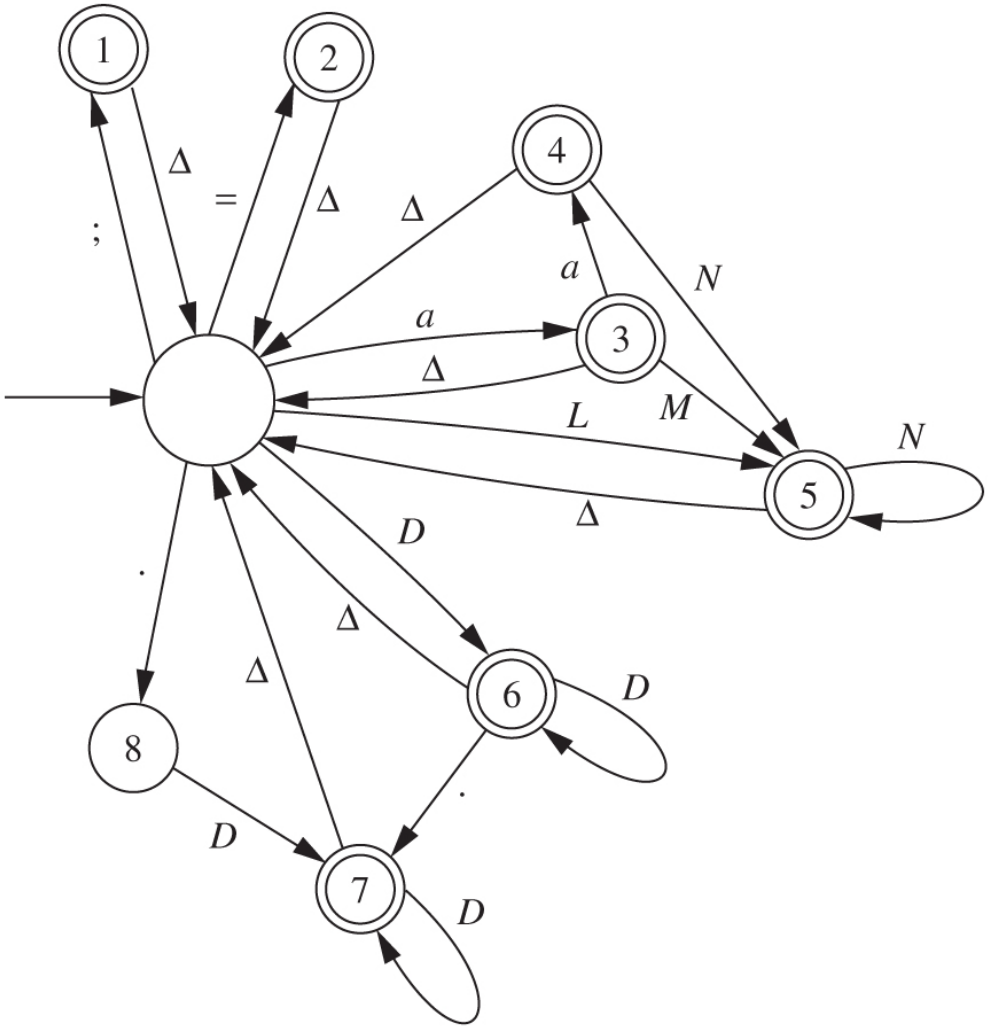


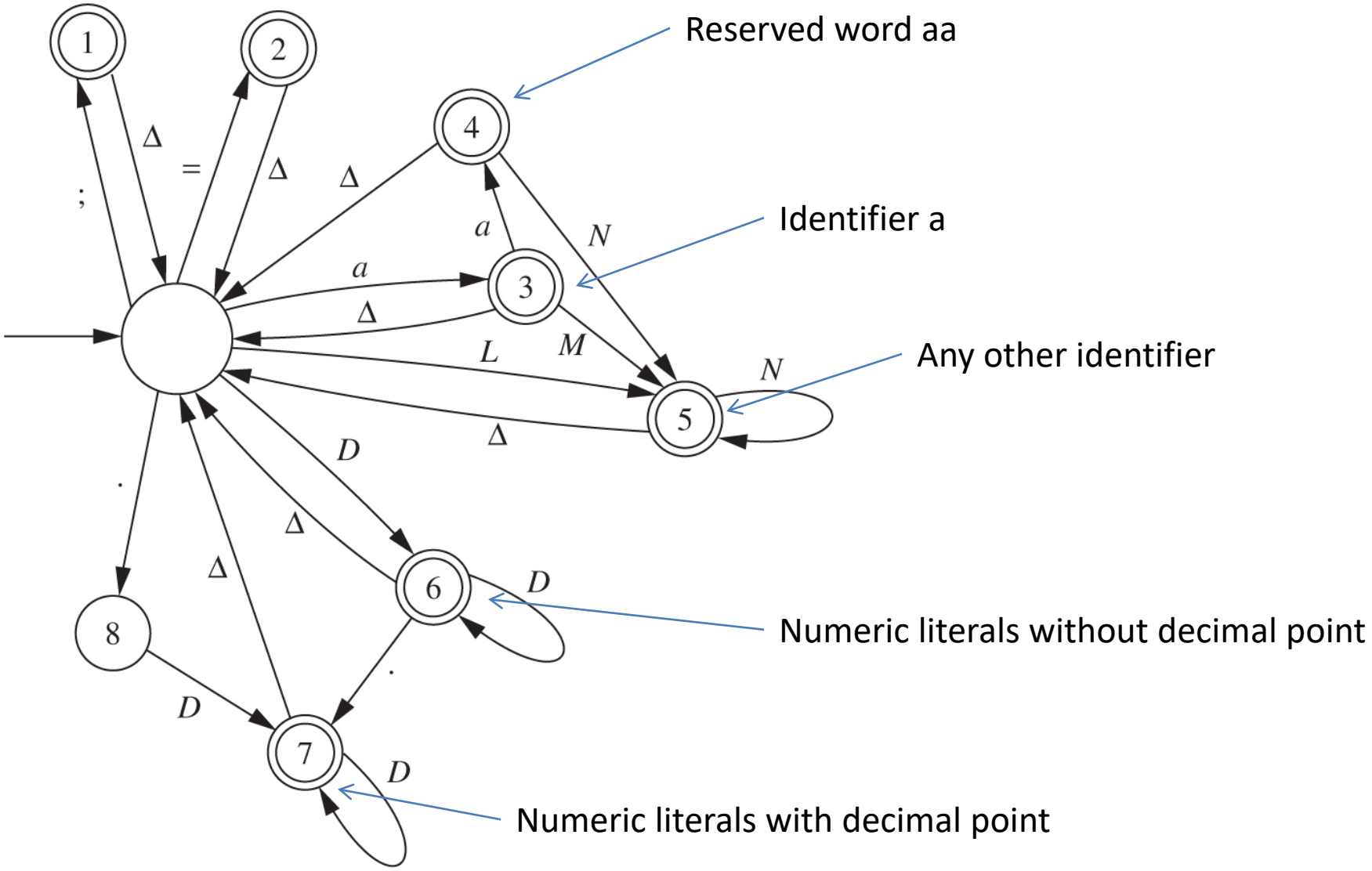
Diagrams are taken from BYJU'S The Learning App

Finite Automata: Lexical Analysis Example

- FAs are ideally suited for lexical analysis, the first stage in compiling a computer program
- A lexical analyzer takes a string of characters and provides a string of "tokens" (indecomposable units)
- Tokens have a simple structure: e.g., "41.3", "main", "="
- The next slide shows an FA that accepts tokens for a simple language based on C
  - The only tokens are identifiers, semicolons, =, *aa*, and numeric literals; tokens are separated by spaces
  - Accepting states represent scanned tokens; each accepting state represents a category of token

- The input alphabet contains the 26 lowercase letters, the 10 digits, a semicolon, an equals sign, a decimal point, and the blank space .
- *D* is any digit
- *L* is a lowercase letter other than *a*
- *M* is *D* or *L*
- *N* is *D* or *L* or *a*
- Δ is a space
- All transitions not shown explicitly go to an error state and stay there




1
2
3
4
5
6
7
8
;
Δ
=
Δ
Δ
a
N
a
Δ
L
M
N
Δ
D
.
Δ
Δ
D
8
D
7
D
.
Reserved word aa
Identifier a
Any other identifier
Numeric literals without decimal point
Numeric literals with decimal point

Definition (Finite Automata) A finite automaton (FA) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where

- Q is a finite set of states;
- Σ is a finite input alphabet;
- $q_0 \in Q$ is the initial state;
- $A \subseteq Q$ is the set of accepting states;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

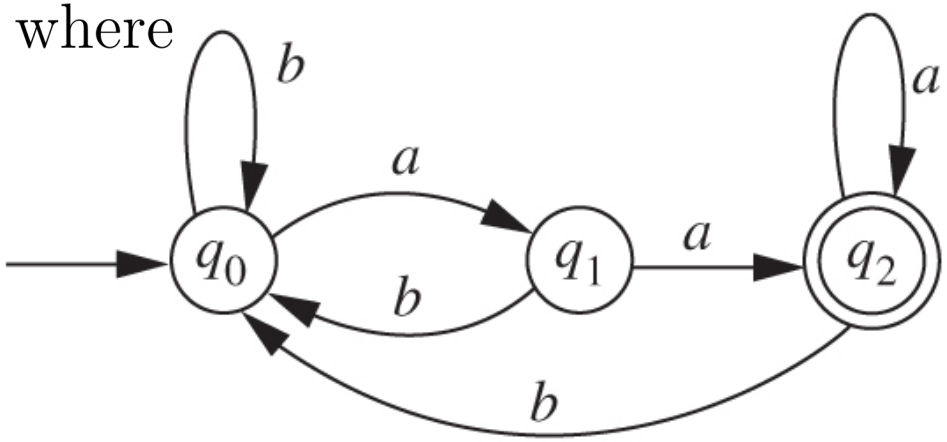
For any element $q \in Q$ and any symbol $\sigma \in \Sigma$, we interpret $\delta(q, \sigma)$ as the state to which the FA moves, if it is in state q and receives the input σ.

Definition (Finite Automata) A finite automaton (FA) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where

- Q is a finite set of states;
- Σ is a finite input alphabet;
- $q_0 \in Q$ is the initial state;
- $A \subseteq Q$ is the set of accepting states;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

Example: $M = (Q, \Sigma, q_0, A, \delta)$, where

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $A = \{q_2\}$



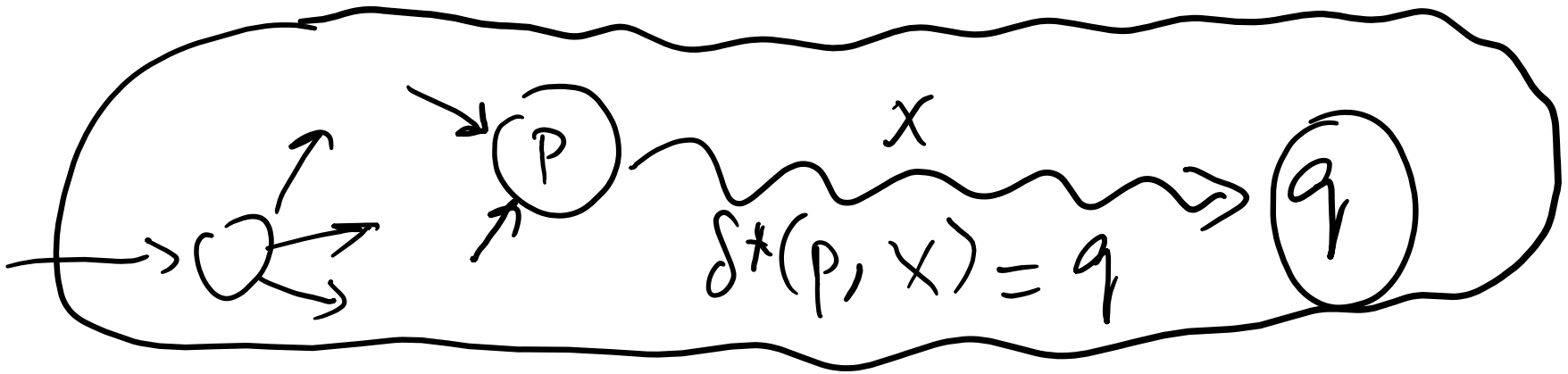
$\delta(q_0, a) = q_1;\ \delta(q_0, b) = q_0;\ \delta(q_1, a) = q_2;\ \delta(q_1, b) = q_0;$
$\delta(q_2, a) = q_2;\ \delta(q_2, b) = q_0;$

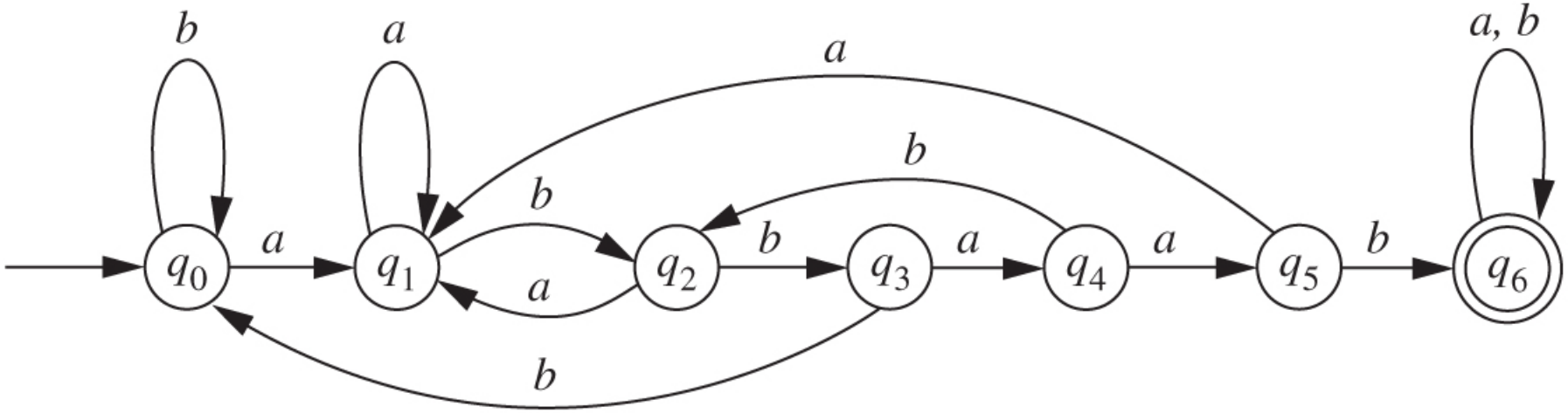
Definition (The Extended Transition Function δ^*)
Let $M = (Q, \Sigma, q_0, A, \delta)$ be FA. We define the extended transition function

$$\delta^* : Q \times \Sigma^* \to Q$$

- For every $q \in Q$, $\delta^*(q, \Lambda) = q$
- For every $q \in Q$, every $y \in \Sigma^*$, and every $\sigma \in \Sigma$,

$$\delta^*(q, y\sigma) = \delta(\delta^*(q, y), \sigma).$$

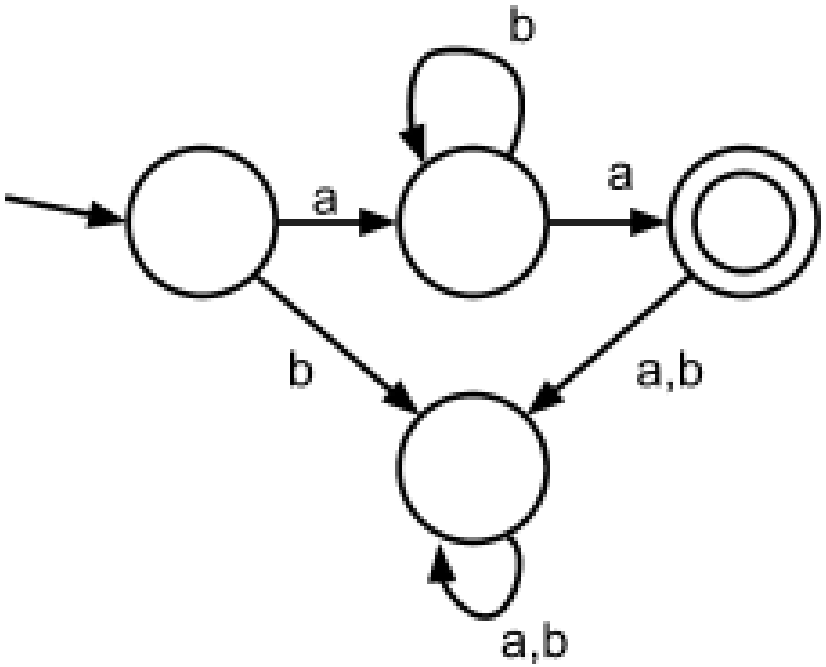




Evaluation of $\delta^*(q_0, baa)$

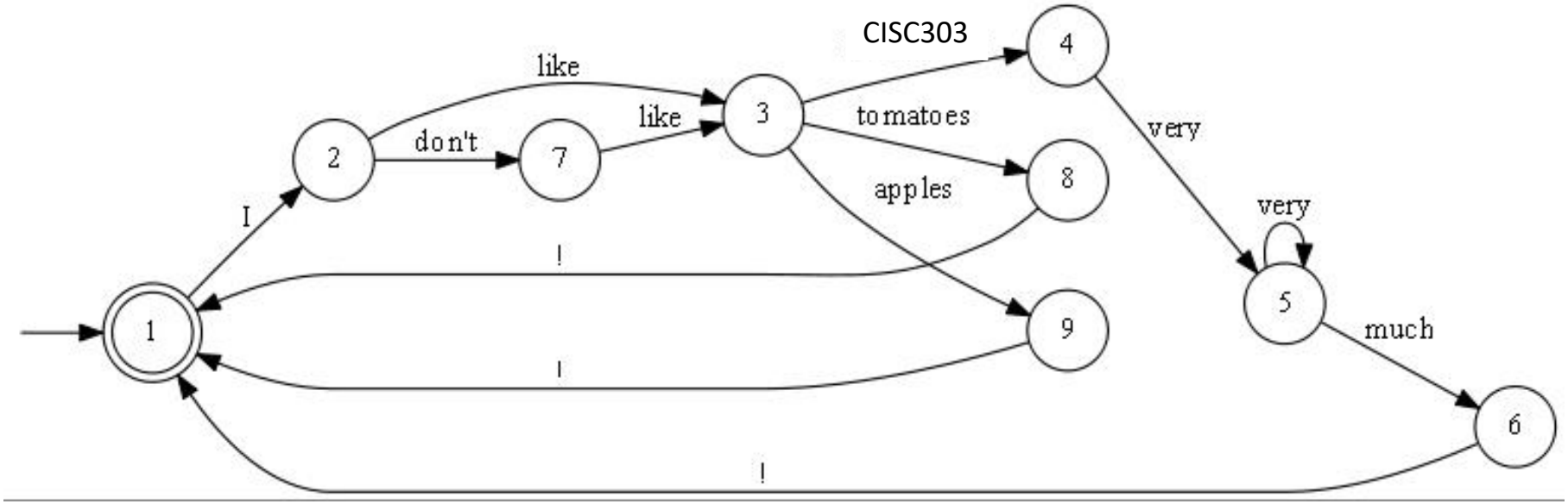
- $\delta^*(q_0, baa) = \delta(\delta^*(q_0, ba), a) = \delta(\delta(\delta^*(q_0, b), a), a)$
  $= \delta(\delta(\delta^*(q_0, \Lambda b), a), a)$
  $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), b), a), a)$
  $= \delta(\delta(\delta(q_0, b), a), a) = \delta(\delta(q_0, a), a)$
  $= \delta(q_1, a) = q_1$

What language is accepted by this finite automaton?



The language of all strings that have exactly two letters “a”:
at the beginning and at the end of the string

What language is accepted by this finite automaton?



The alphabet is

- I
- like
- don’t
- CISC303
- tomatoes
- apples
- very
- much
- !

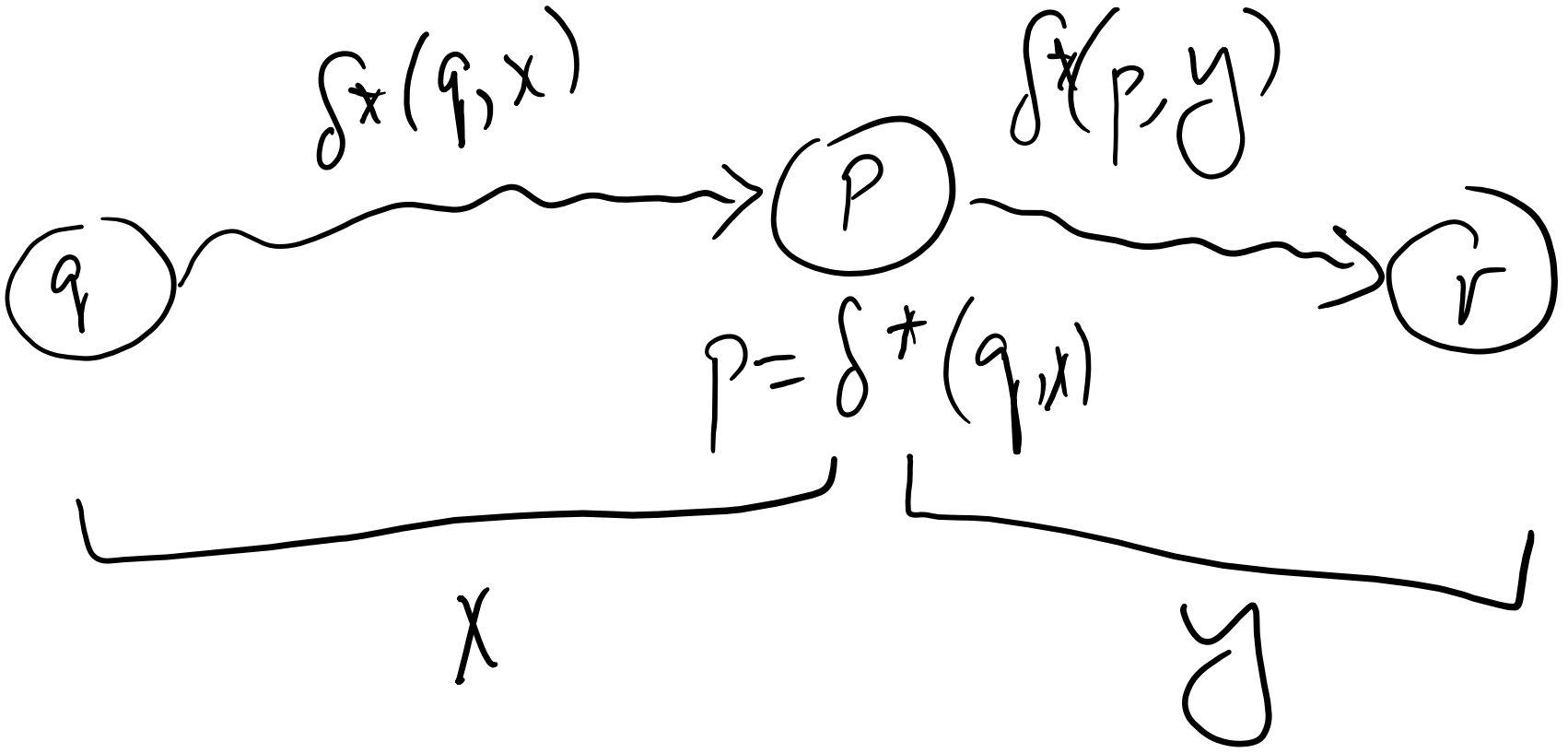
Add missing transitions; The language contains all chains (without spaces) of

- I like apples!
- I don’t like apples!
- I like tomatoes!
- I don’t like tomatoes!
- I like CISC303 very very* much!
- I don’t like CISC303 very very* much!

Can you propose a smaller FA that accepts the same language?

Merge states 8, 9 and 6

Claim. *For every* $x, y \in \Sigma^*$ $\quad \delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$



Claim. *For every* $x, y \in \Sigma^*$ $\quad \delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$

Proof. For every $y \in \Sigma^*$ we need to prove that $P(y)$ is true, where $P(y)$ is the statement "for every $x \in \Sigma^*$, $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$".

- BS: $\forall x \in \Sigma^* \ \delta^*(q, x\Lambda) = \delta^*(\delta^*(q, x), \Lambda)$. This is true beacause $\delta^*(\delta^*(q, x), \Lambda) = \delta^*(q, x)$, i.e., we have $\delta^*(q, x\Lambda) = \delta^*(q, x)$.
- IH: $y \in \Sigma^*$, and $\forall x \in \Sigma^*$, $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$.
- IS: $\forall x \in \Sigma^*$, $\delta^*(q, x(y\sigma)) = \delta^*(\delta^*(q, x), y\sigma)$

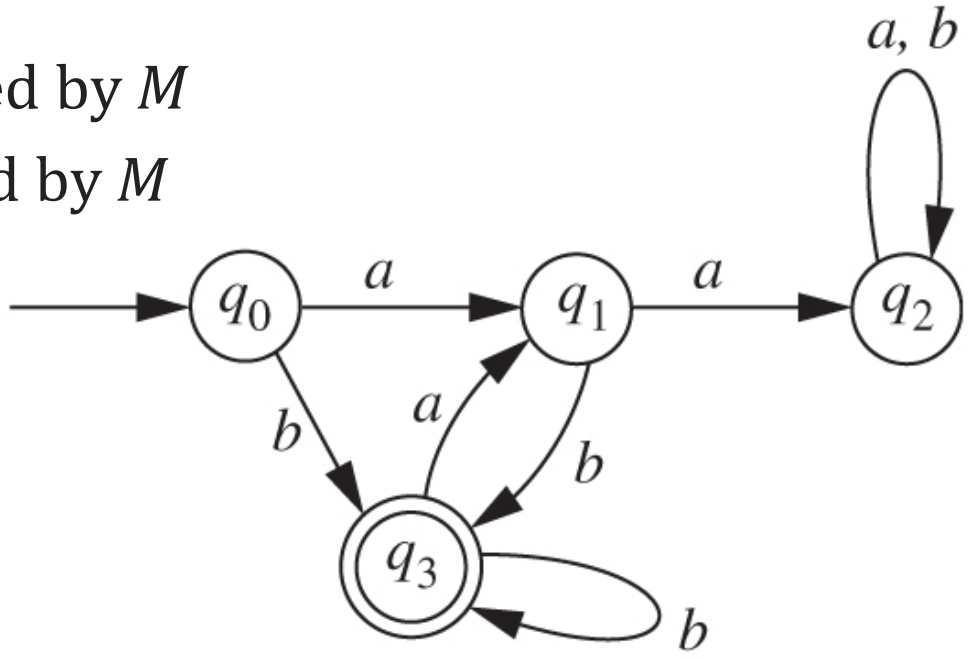
$$\begin{array}{ccccc} \delta^*(q, x(y\sigma)) & = & \delta^*(q, (xy)\sigma) & = & \\ \delta(\delta^*(q, xy), \sigma) & = & \delta(\delta^*(\delta^*(q, x), y), \sigma) & = & \\ \delta^*(\delta^*(q, x), y\sigma) & & & & \end{array}$$

□

- Definition:
  - Let $M=(Q, \Sigma, q_0, A, \delta)$ be an FA, and let $x \in \Sigma^*$. Then x is *accepted* by M if $\delta^*(q_0, x) \in A$ and *rejected* otherwise

babbbbabb is accepted by *M*

bbbbaaaab is rejected by *M*



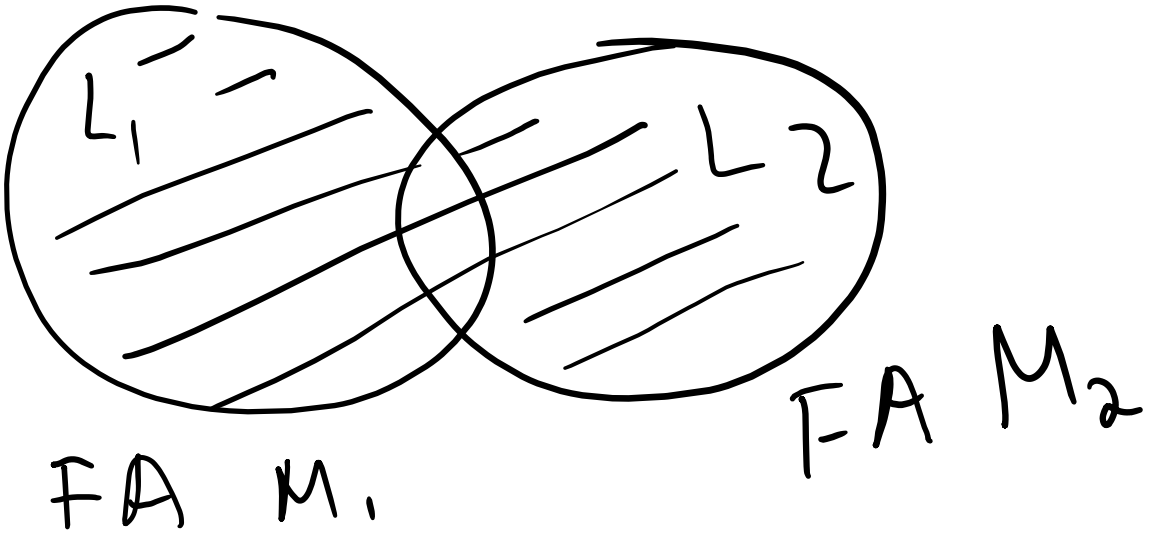
- The *language* accepted by *M* is

  $L(M) = \{x \in \Sigma^* \mid x \text{ is accepted by } M\}$

Accepting the Union, Intersection, or Difference of Two Languages

- Suppose that L_1 and L_2 are languages over Σ
  - Given an FA that accepts L_1 and another FA that accepts L_2, we can construct one that accepts $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$

  How to construct an FA for $L_1 \cup L_2$?



Accepting the Union, Intersection, or Difference of Two Languages

- Suppose that L_1 and L_2 are languages over Σ
  - Given an FA that accepts L_1 and another FA that accepts L_2, we can construct one that accepts $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$

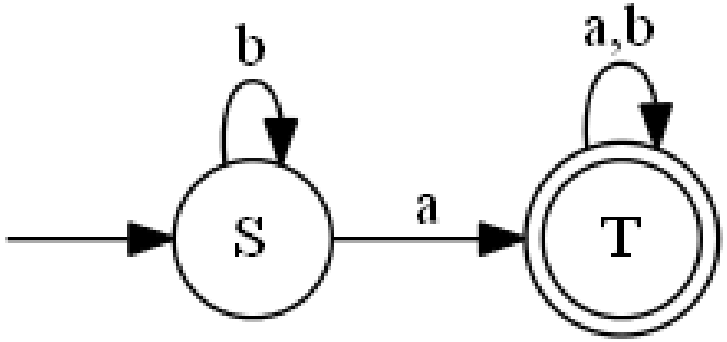
  How to construct an FA for $L_1 \cup L_2$?

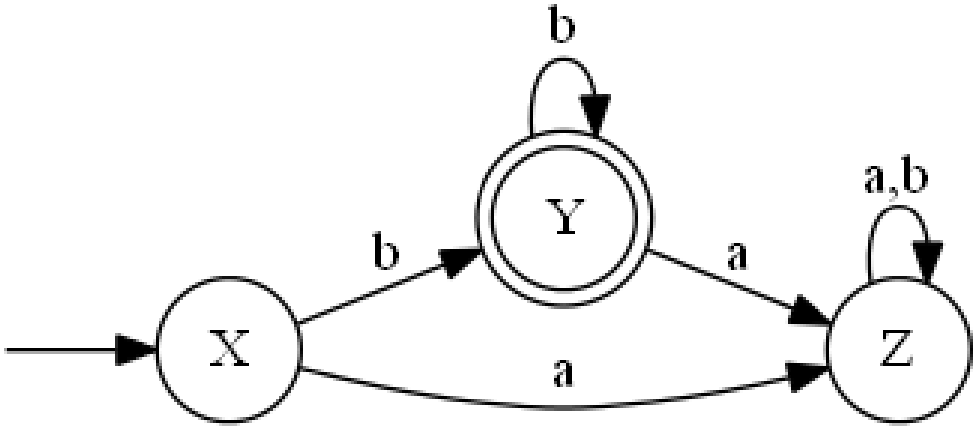
  - The idea is to construct an FA that executes both of the original FAs at the same time
  - This works because if $x \in \Sigma^*$, then knowing whether $x \in L_1$ and whether $x \in L_2$ is enough to determine whether $x \in L_1 \cup L_2$

Accepting the Union, Intersection, or Difference of Two Languages (cont'd.)

- Theorem: Suppose $M_1=(Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2=(Q_2, \Sigma, q_2, A_2, \delta_2)$ are FAs accepting L_1 and L_2. Let $M=(Q, \Sigma, q_0, A, \delta)$ be defined as follows:
  - $Q = Q_1 \times Q_2$
  - $q_0 = (q_1, q_2)$
  - $\delta((p, q), \sigma) = (\delta_1(p, \sigma), \delta_2(q, \sigma))$
- Then, if :
  - $A = \{(p, q) \mid p \in A_1 \text{ or } q \in A_2\}$, M accepts $L_1 \cup L_2$
  - $A = \{(p, q) \mid p \in A_1 \text{ and } q \in A_2\}$, M accepts $L_1 \cap L_2$
  - $A = \{(p, q) \mid p \in A_1 \text{ and } q \notin A_2\}$, M accepts $L_1 - L_2$

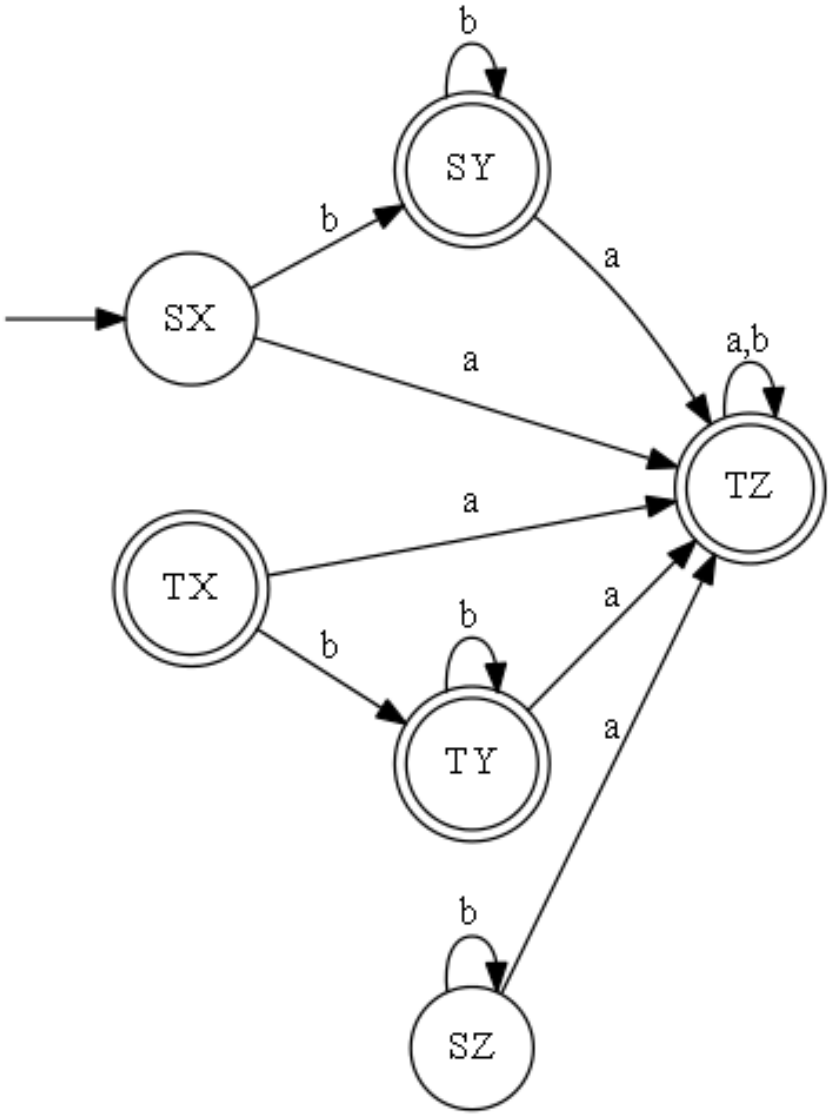
L1 = all strings that include “a”





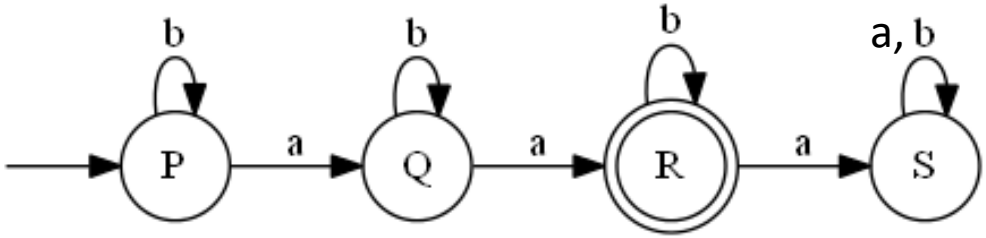
L2 = all strings that do not include “a”

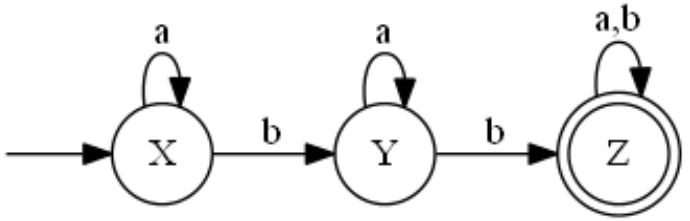
Union of L1 and L2



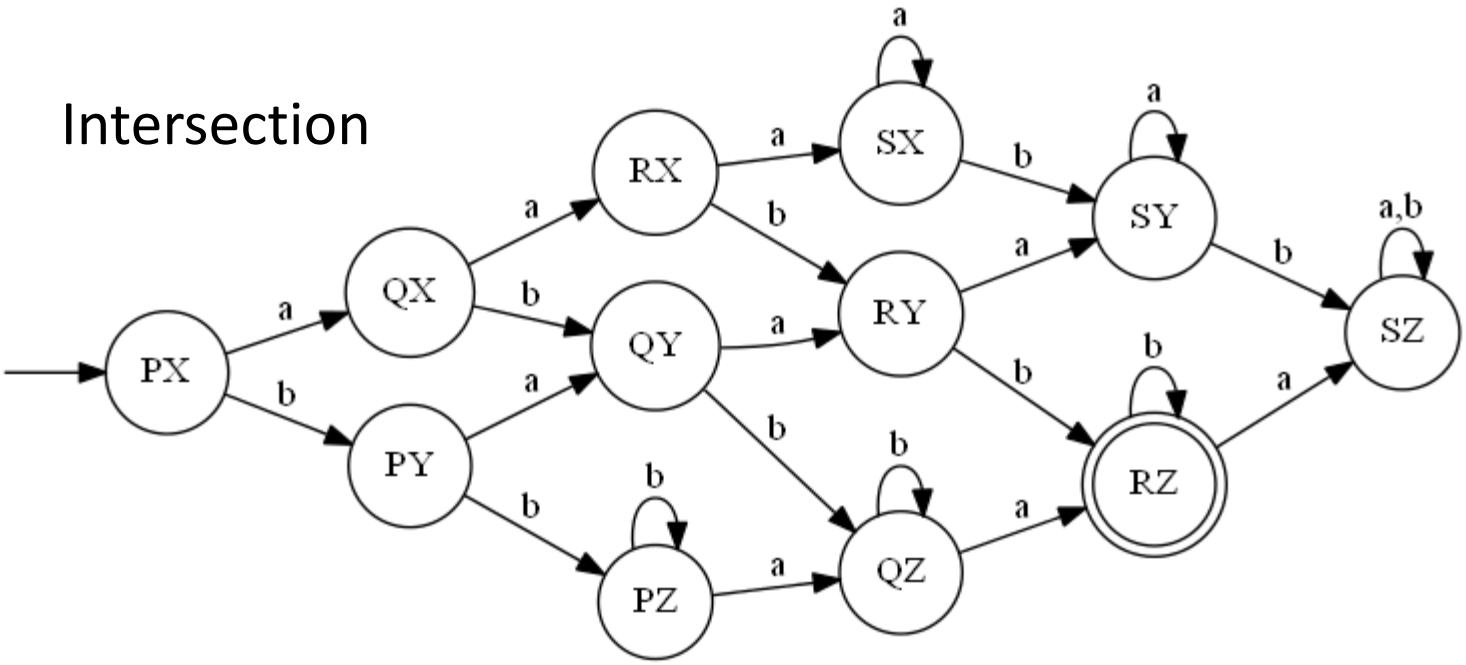
Construct an FA to accept strings with exactly 2 a's and at least 2 b's

Strings with exactly 2 a's ∩ Strings with at least 2 b's

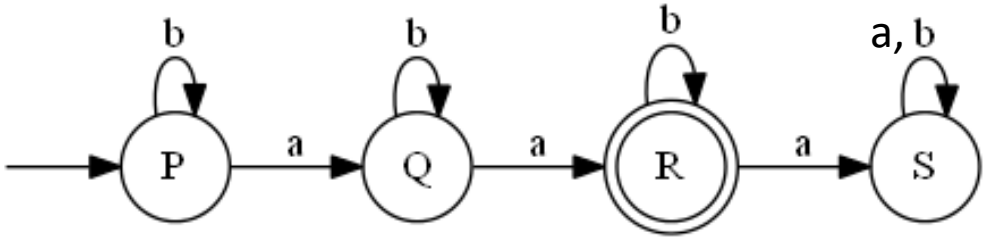




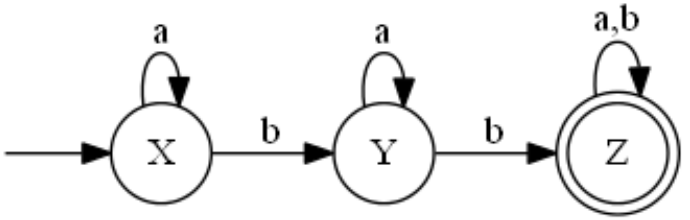
Intersection



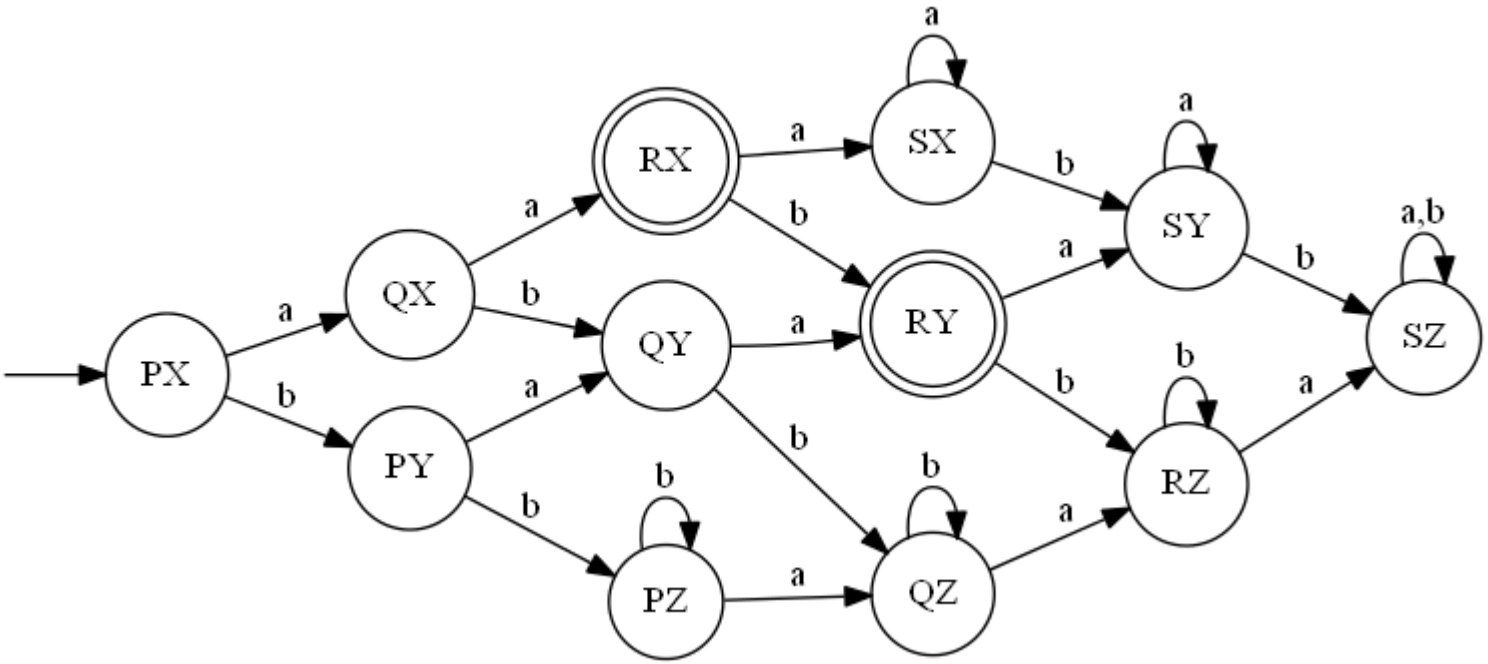
Strings with exactly 2 a's



Strings with at least 2 b's



Difference, i.e., strings with exactly 2 a's and at most 1 b



We prove the first part of the theorem.

- Theorem: Suppose $M_1=(Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2=(Q_2, \Sigma, q_2, A_2, \delta_2)$ are FAs accepting L_1 and L_2. Let $M=(Q, \Sigma, q_0, A, \delta)$ be defined as follows:
  - $Q = Q_1 \times Q_2$
  - $q_0 = (q_1, q_2)$
  - $\delta((p, q), \sigma) = (\delta_1(p, \sigma), \delta_2(q, \sigma))$

then, if $A = \{(p, q) \mid p \in A_1 \text{ or } q \in A_2\}$, M accepts $L_1 \cup L_2$.

Proof sketch. At any point during the operation of M, if (p, q) is the current state, then p, and q are the current states of M_1, and M_2. This follows from

Prove by structural induction at home

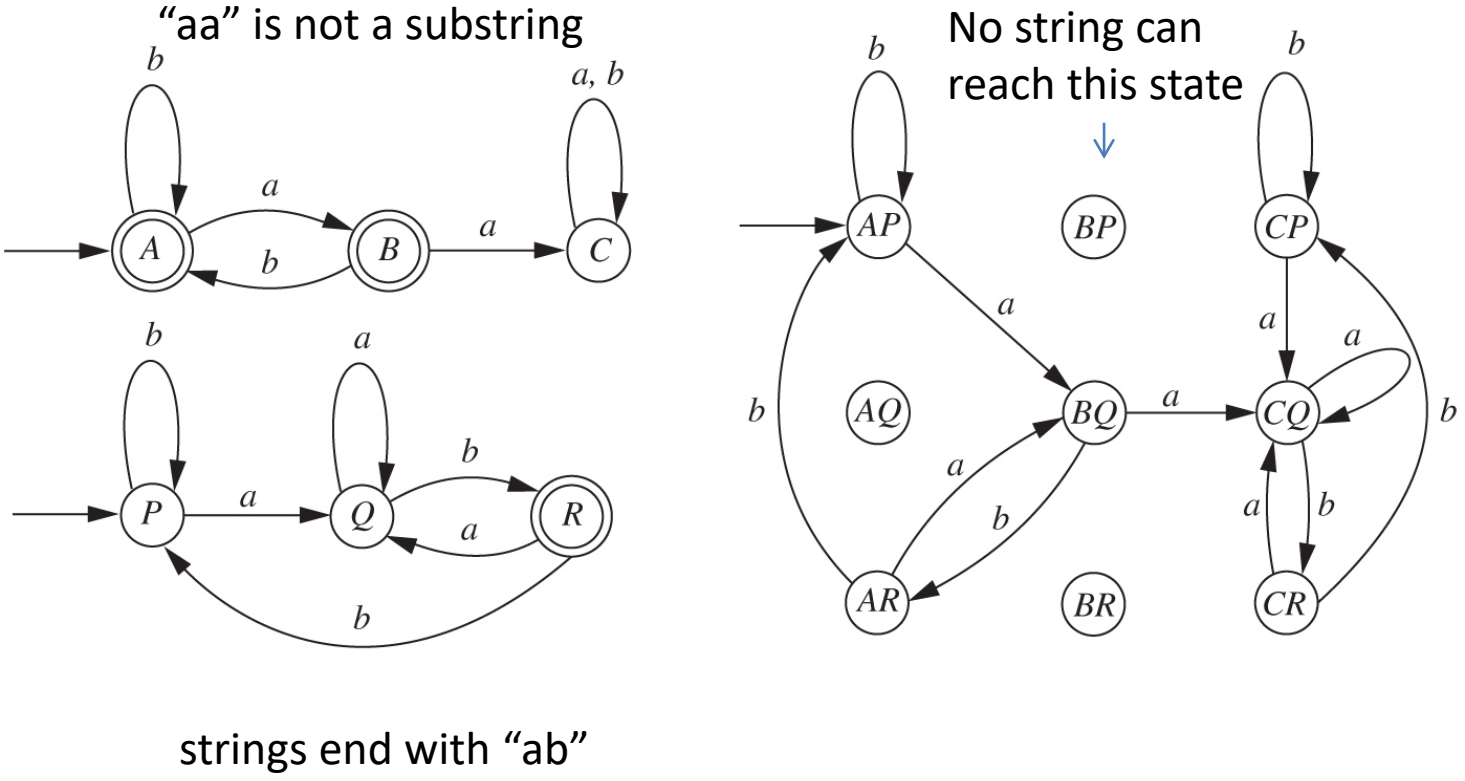
$$\delta^*(q_0, x) = (\delta_1^*(q_1, x), \delta_2^*(q_2, x)) \quad \forall x \in \Sigma^*.$$

For every $x \in \Sigma^*$, x is accepted by M iff $\delta^*(q_0, x) \in A$ but according to defs of A and δ^*, this is true if $\delta_1^*(q_1, x) \in A_1$ or $\delta_2^*(q_2, x) \in A_2$, i.e., if $x \in L_1 \cup L_2$. □

Homework: Make sure you learn how to prove the full version of this theorem including the structural induction, union, intersection and difference (see textbook). Don’t submit!

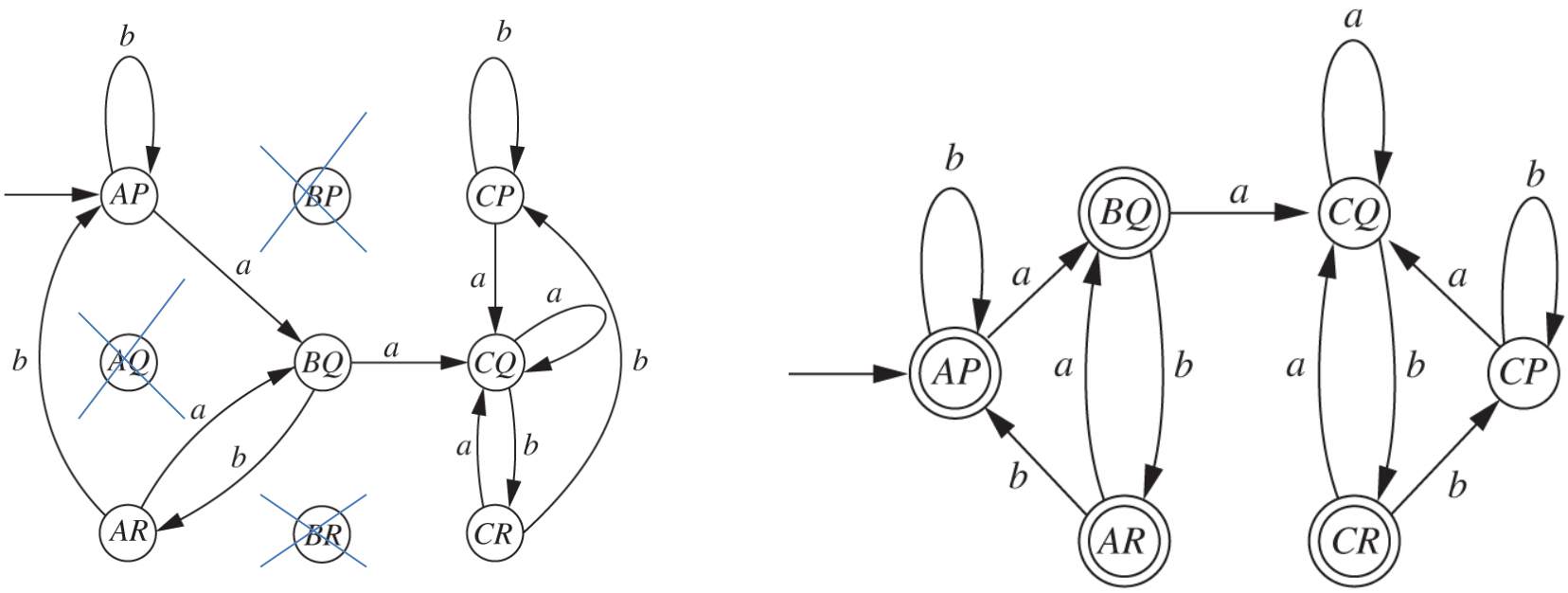
Accepting the Union, Intersection, or Difference of Two Languages (cont'd.)

- Given two machines, create the Cartesian product of the state sets, and draw the necessary transitions



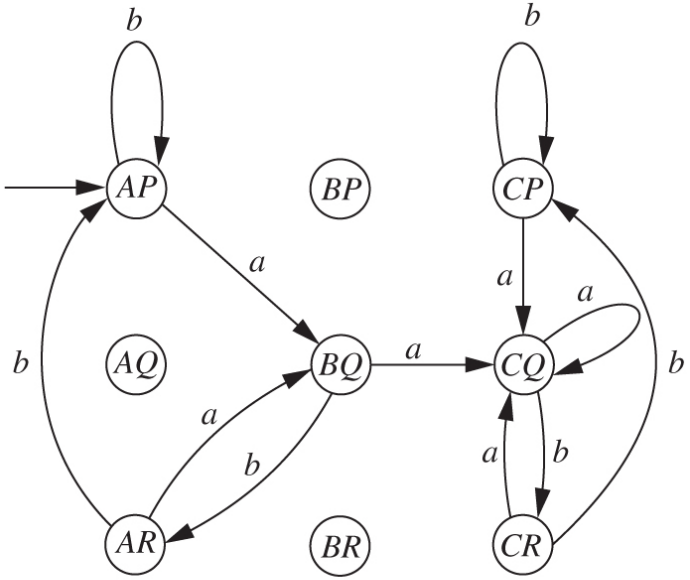
Accepting the Union, Intersection, or Difference of Two Languages (cont'd.)

- Simplify the resulting machine, if possible, and designate the appropriate accepting states
- The machine below accepts the **union** of the two languages

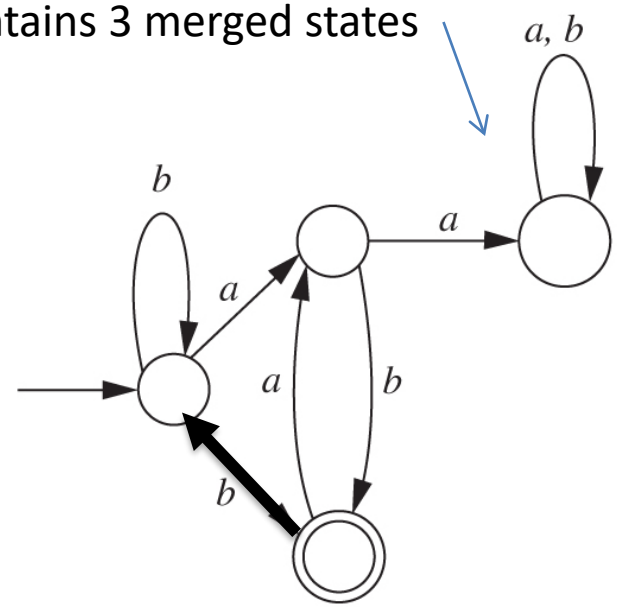


Accepting the Union, Intersection, or Difference of Two Languages (cont'd.)

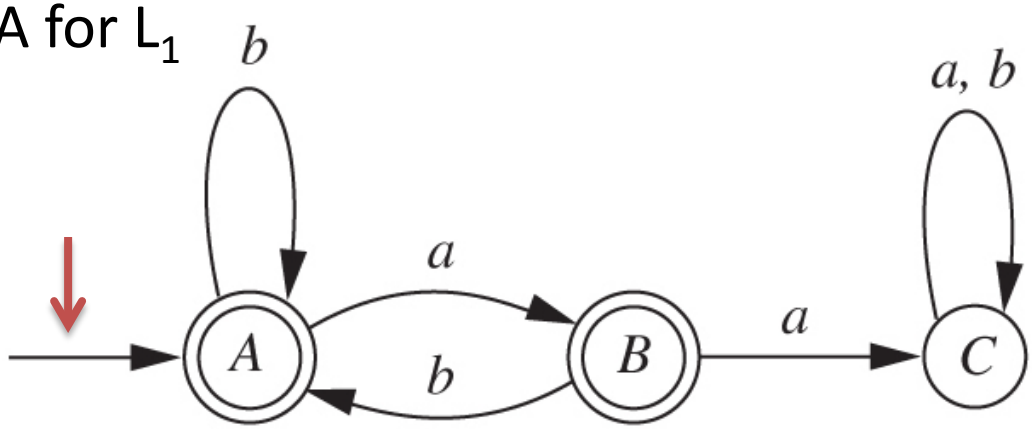
- For the **intersection**, we can simplify further, and we end up with the machine on the right
- The simplification involved turning states CP, CQ, and CR into a single state (none of them was accepting, and there was no way to leave them)

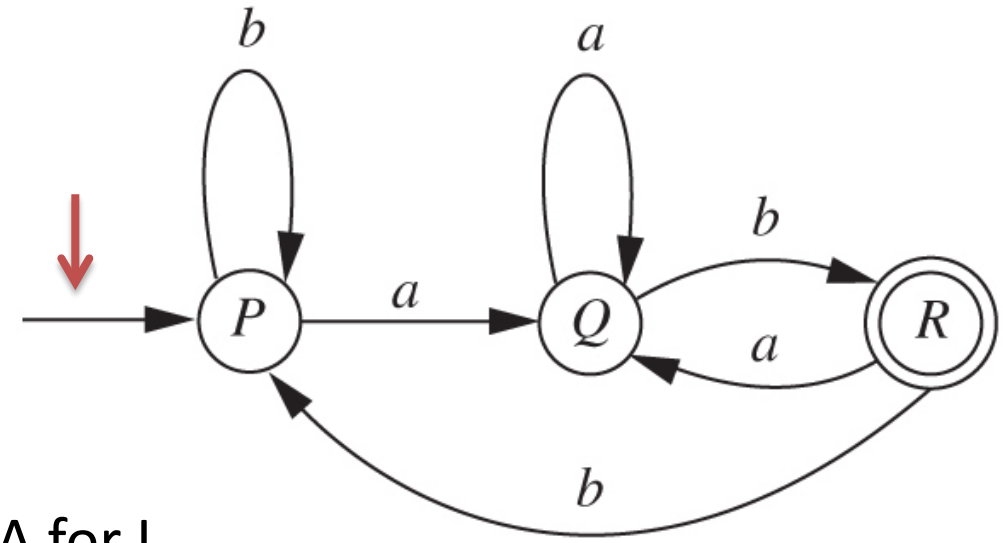


Contains 3 merged states



FA for L_1





FA for L_2

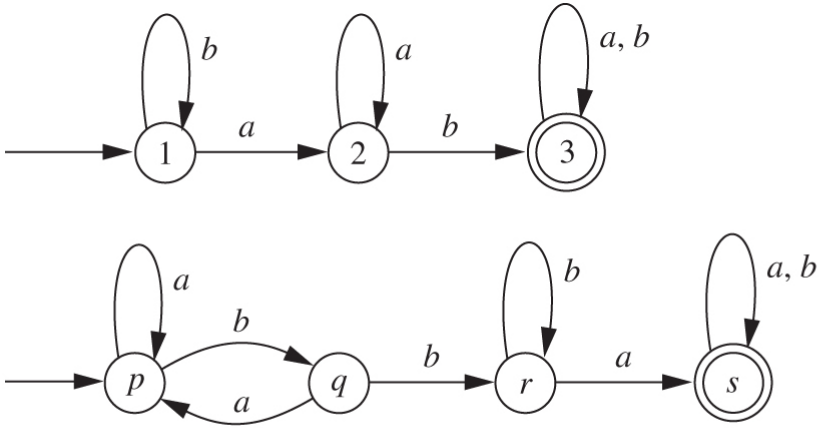
(a)

Input: abbaa

		$L_1 \cup L_2$	$L_1 \cap L_2$	$L_1 - L_2$
_	AP	y	n	y
a	BQ	y	n	y
b	AR	y	y	n
b	AP	y	n	y
a	BQ	y	n	y
a	CQ	n	n	n

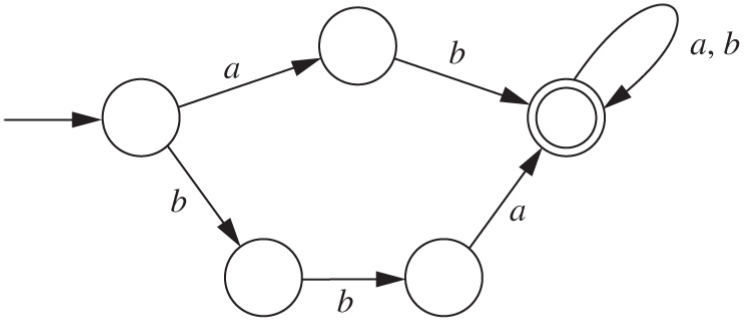
Example: FA accepting strings that contain either “ab” or “bba”

x contains “ab”

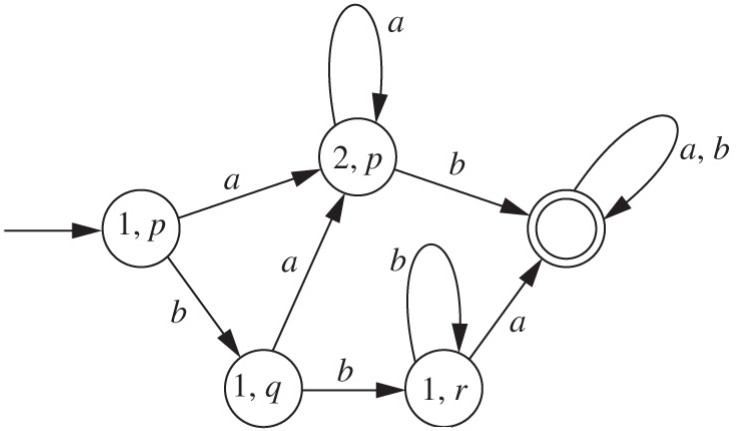


x contains “bba” (a

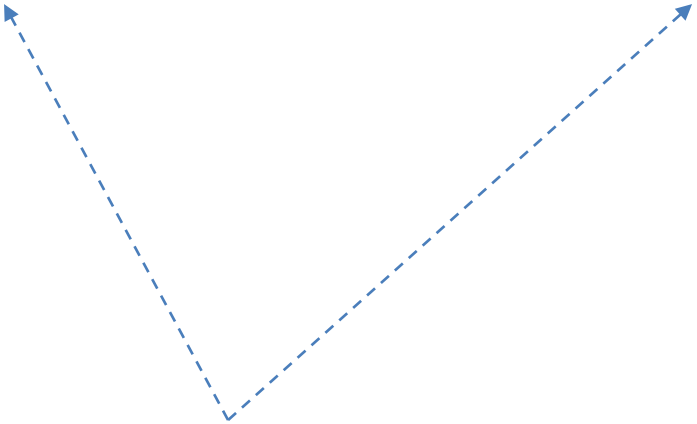
We can try to build FA with 12 states but we can do less with “if”-like fork



(b)

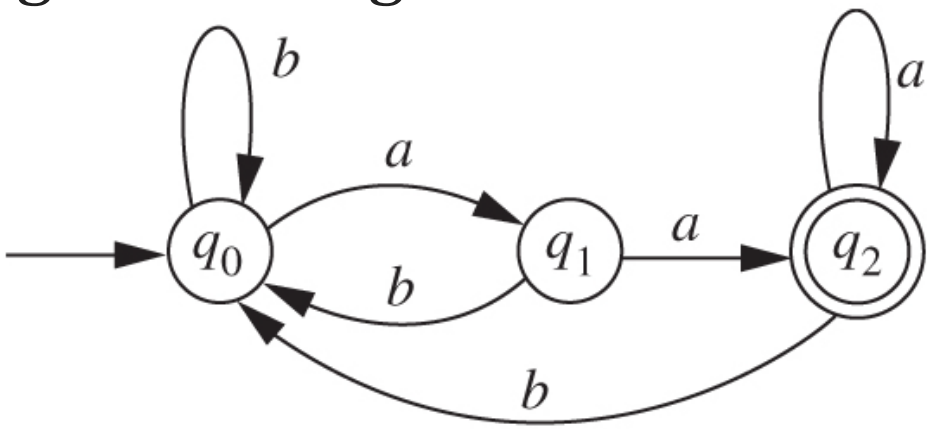


- Can we construct FA with fewer states?
- Can we be sure that this number of states is enough?
- We will study the connection between the "complexity" of input and the "complexity" of algorithm.



Note that this is not a traditional notion of algorithm complexity in the theory of computer science that we will study later

Distinguishing One String from Another



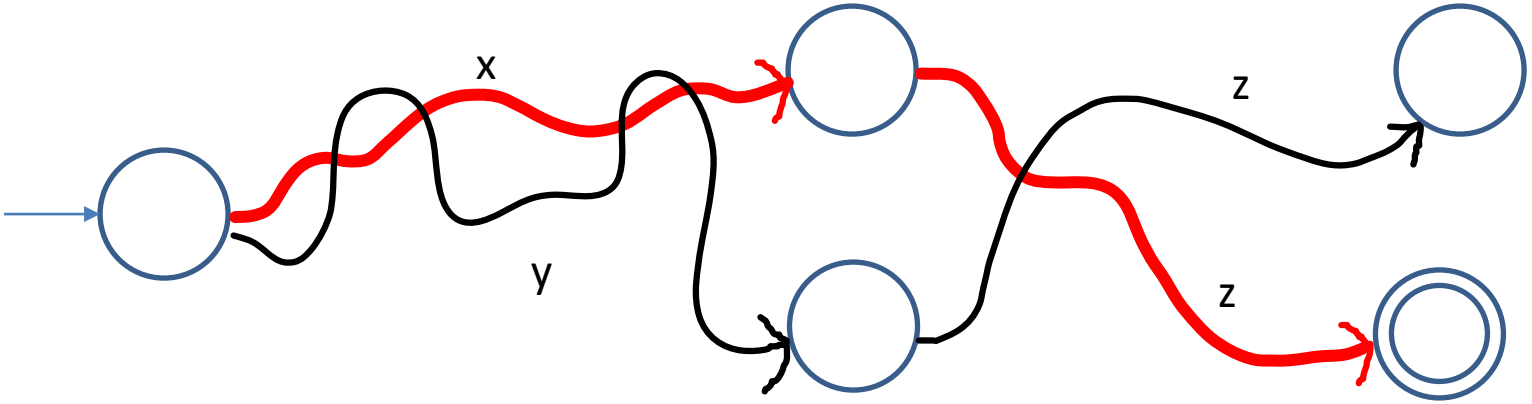
Strings ending in *aa*

- Any FA, ignores, or "forgets", a lot of information
- An FA doesn't remember which string has been seen
  - *aba* and *aabbabbabaaaba* lead to the same state;
  - *aba* and *ab*, however, lead to different states; the essential difference is that one ends with *a* and the other doesn't
  - *aba* and *ab* are *distinguishable* with respect to the language accepted by the FA; there is at least one string *z* (such as *a*) so that *abaz* is in the language (i.e., is accepted) and *abz* is not, or vice versa

Distinguishing One String from Another (cont'd.)

- Definition:
  - If L is a language over Σ, and $x, y \in \Sigma^*$, then x and y are *L-distinguishable*, if there is a string $z \in \Sigma^*$ such that

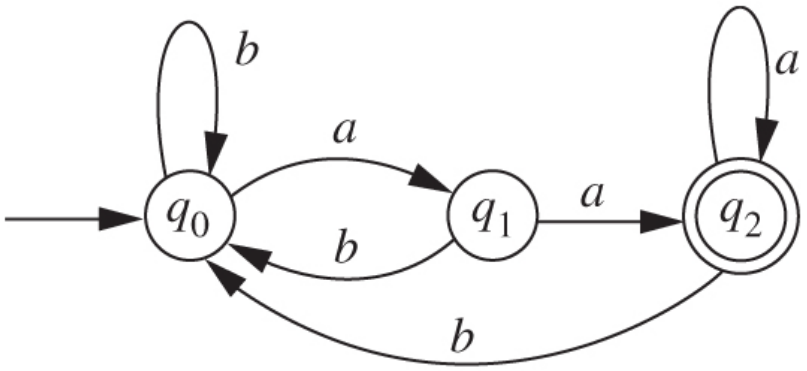
    either $xz \in L$ and $yz \notin L$, or $xz \notin L$ and $yz \in L$
  - A string z having this property is said to distinguish x and y with respect to L
  - Equivalently, x and y are L-distinguishable if $L/x \neq L/y$, where $L/x = \{z \in \Sigma^* \mid xz \in L\}$



Distinguishing One String from Another (cont'd.)

- **Theorem**: Suppose $M=(Q, \Sigma, q_0, A, \delta)$ is an FA accepting $L \subseteq \Sigma^*$
  - If $x, y \in \Sigma^*$ are L-distinguishable, then $\delta^*(q_0, x) \neq \delta^*(q_0, y)$
  - For all $n \geq 2$, if there is a set of n pairwise L-distinguishable strings in Σ^*, then Q must contain **at least** n states

This shows why we need at least three states in any FA that accepts the language L of strings ending in aa: $\{\Lambda, a, aa\}$ contains 3 pairwise L-distinguishable strings



Part I: If x and y are two strings in Σ^* that are L-distinguishable, then $\delta^*(q_0, x) \neq \delta^*(q_0, y)$

Proof sketch. x, and y are L-distinguishable $\Rightarrow$
$\exists z \in \Sigma^*$ s.t. $xz \in L$, and $yz \notin L$ (or vice versa). Because M accepts L then either $\delta^*(q_0, xz) \in A$ and $\delta^*(q_0, yz) \notin A$ (or vice versa), i.e.,

$$\delta^*(q_0, xz) \neq \delta^*(q_0, yz).$$

However,

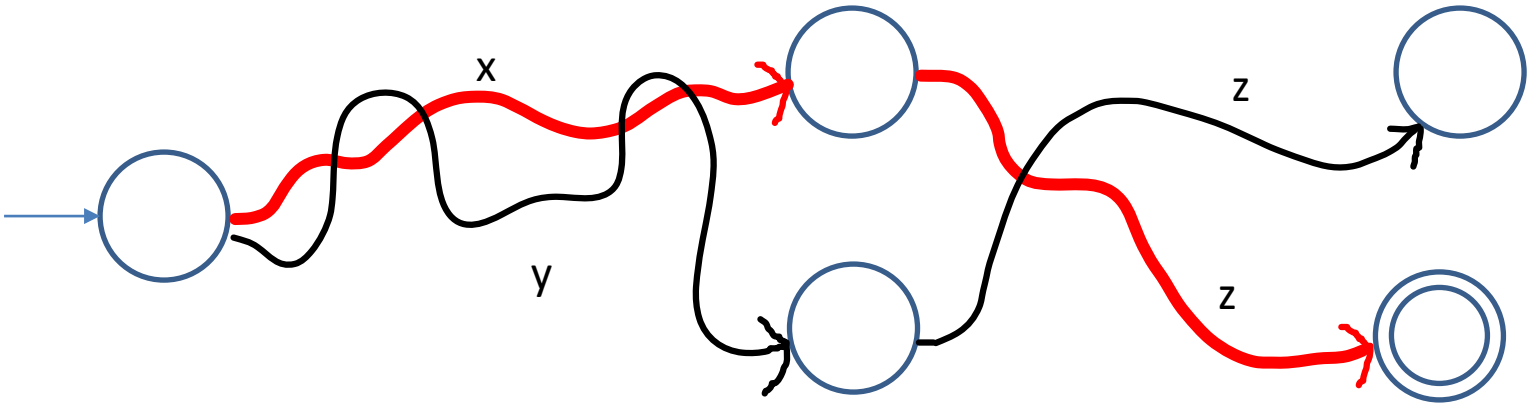
$$\begin{aligned} \delta^*(q_0, xz) = & \qquad \delta^*(\delta^*(q_0, x), z) \\ \delta^*(q_0, yz) = & \qquad \delta^*(\delta^*(q_0, y), z). \end{aligned}$$

Because the left sides are different, the right sides must be also, and then

$$\delta^*(q_0, x) \neq \delta^*(q_0, y).$$

□

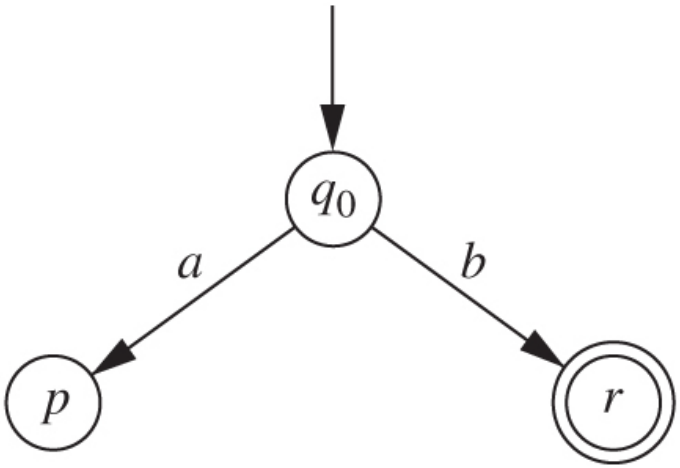
Part II: For all $n \geq 2$, **if** there is a set of n pairwise L-distinguishable strings in Σ^*, **then** Q must contain at least n states

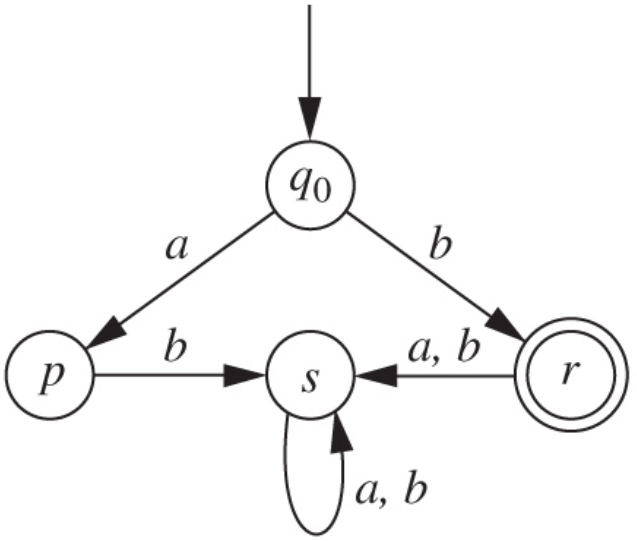


Proof sketch. The second part of the theorem follows from the first. If M had fewer than n states, then at least two of the n strings would cause M to end up in the same state. This is contradiction because these strings are L-distinguishable. □

Distinguishing One String from Another (cont'd.)

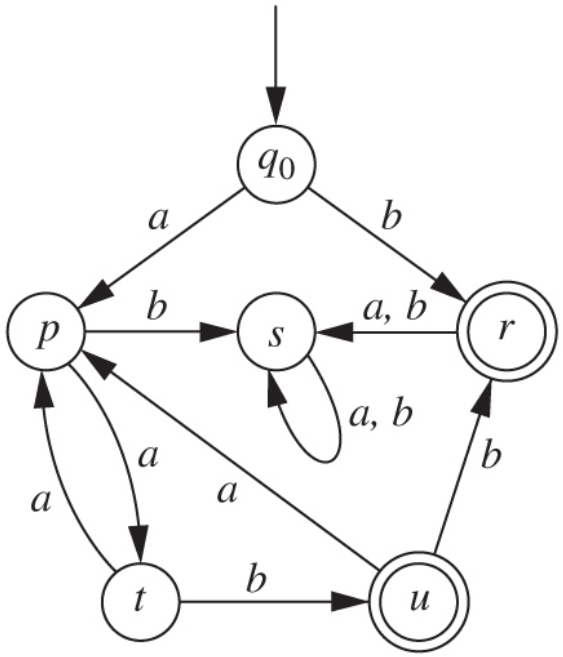
- To create an FA to accept $L = L_1L_2 = \{aa, aab\}^*\{b\}$, we notice first that Λ,a $\notin L$, $b \in L$, and Λ,b, and a are L-distinguishable (for example, $\Lambda b \in L$, $ab \notin L$)
  - We need at least the states in the first diagram
  - L contains b but nothing else that begins with b, so we add a state s to take care of illegal prefixes
  - If the input starts with aa we, need to leave state p because a and aa are L-distinguishable; create state t





Distinguishing One String from Another (cont'd.)

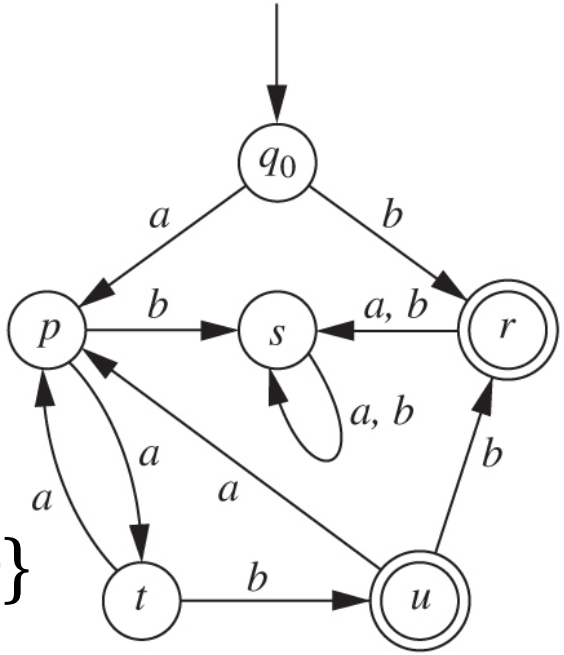
- $\delta(t, b)$ must be accepting, because $aab \in L$ but distinguishable from b; call that new state u
- Let $\delta(u,b)$ be r, because $aabb$ is in L but not a prefix of any other string in L
- States t and u can be thought of as representing the end of an occurrence of aa or aab; if the next symbol is a it's the start of a new occurrence, so go back to p
- The result is shown here



Distinguishing One String from Another (cont'd.)

	Λ	a	b	aab	aa	ab
Λ	-	b	Λ	Λ	bb	b
a		-	Λ	b	b	ab
b			-	aab	bb	Λ
aab				-	bb	Λ
aa					-	b
ab						-

These are strings z's that distinguish between x, and y



FA to accept $L = \{aa, aab\}^*\{b\}$

Theorem. *For every $L \subseteq \Sigma^*$, if there is an infinite set S of pairwise L-distinguishable strings, then L cannot be accepted by FA.*

Proof sketch. If S is infinite then for every n, S has a subset with n elements. If M was FA accepting L, then previous theorem would say that for every n, M would have at least n states. No FA has this property. □

Example. *For every pair of distinct x and y in $\{a,b\}^*$, x, and y are distinguishable wrt PAL.*

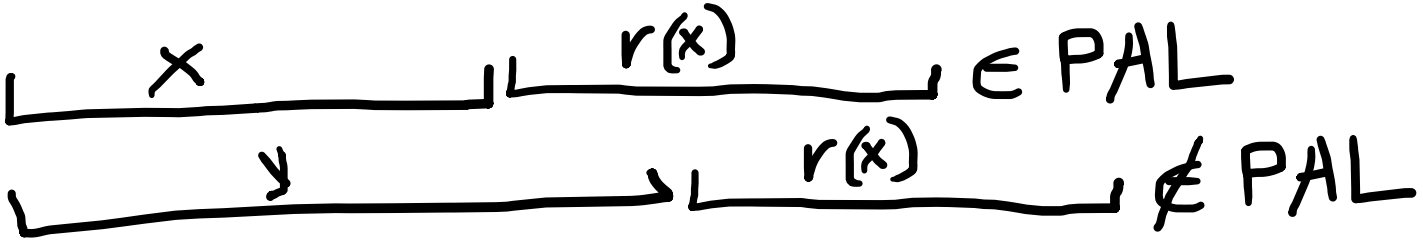
Consider the language of palindromes, PAL, over $\{a,b\}^*$. Let's take two strings x, and y.

- *If $x \neq y$, $|x| = |y|$, then $r(x)$ distinguishes between x, and y, because $xr(x) \in PAL$, and $yr(x) \notin PAL$.*

Example. *For every pair of distinct x and y in $\{a,b\}^*$, x, and y are distinguishable wrt PAL.*

Consider the language of palindromes, PAL, over $\{a,b\}^*$. Let's take two strings x, and y.

- *If $|x| \neq |y|$, w.l.o.g. we assume $|x| < |y|$. If x is not a prefix of y, then $xr(x) \in PAL$, and $yr(x) \notin PAL$.*



Example. *For every pair of distinct x and y in $\{a,b\}^*$, x, and y are distinguishable wrt PAL.*

*. Let's Consider the language of palindromes, PAL, over $\{a,b\}$ take two strings x, and y.

and w, • If $x \neq_{PAL} y$, then $|x| \neq |y|$ then $x(r)$ distinguishes between x because $xr(x) \in PAL$, and $yr(x) \notin PAL$.

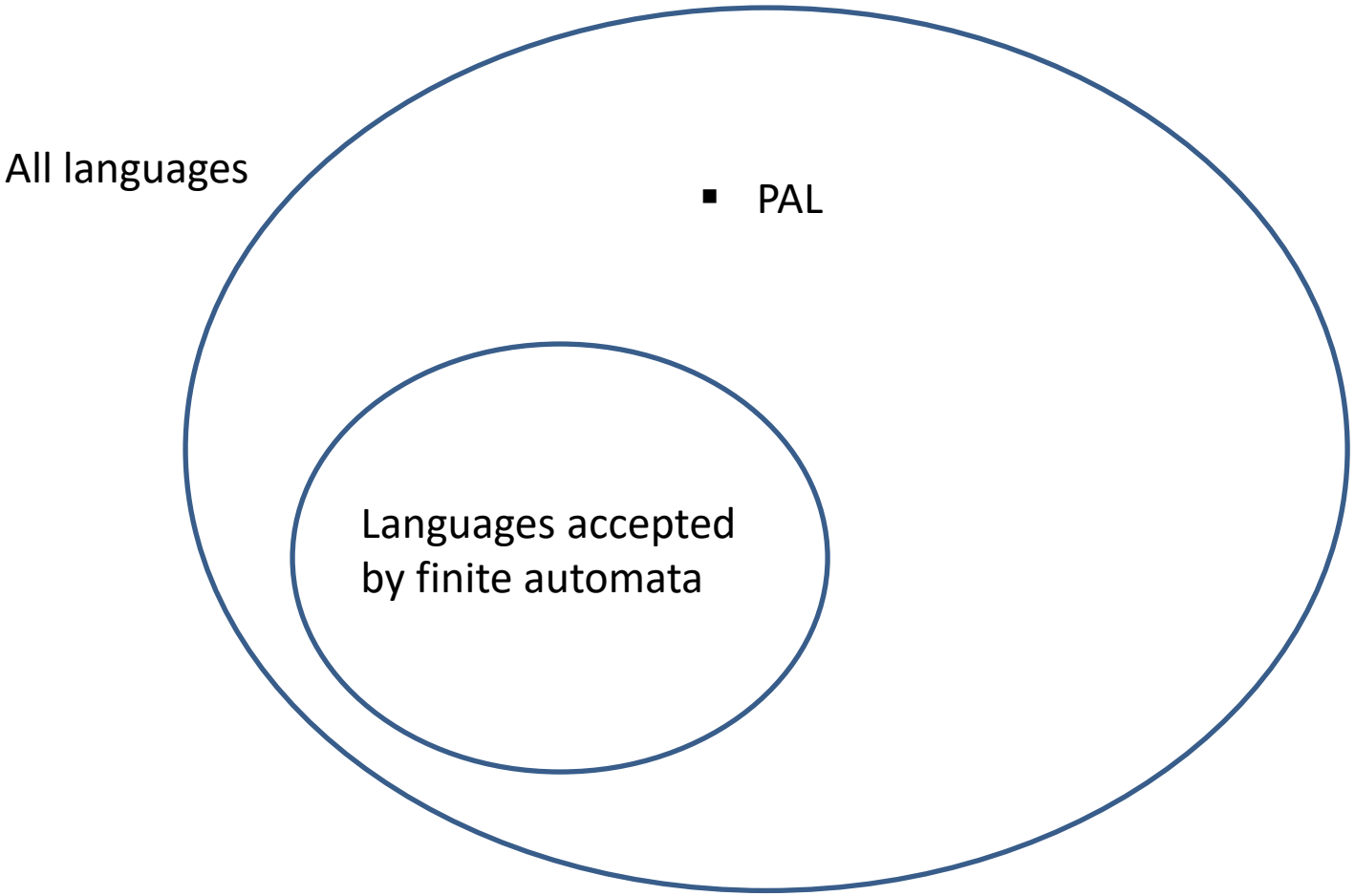
is not a prefix of • If $|x| \neq |y|$, w.l.o.g. we assume $|x| < |y|$. If x AL. y, then $xr(x) \in PAL$, and $yr(x) \notin P$

or some z. If we choose • If x is a prefix of y, then $y = xz$ fo pe then we then symbol σ s.t. that $z\sigma$ is not a palindro

$y\sigma r(x) = xz\sigma r(x) \notin PAL,$ $x\sigma r(x) \in PAL$ *and*

by FA because we can find infinitely many PAL cannot be accepted b gs. (Requires infinite memory to remember PAL-distinguishable string $r(\cdot)$)

So, what do we know about languages …

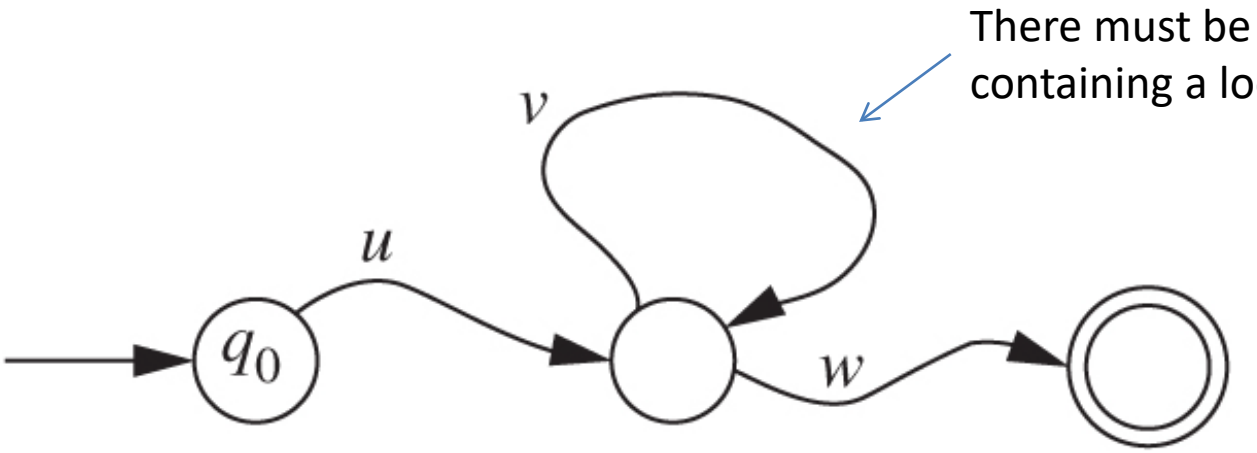


The Pumping Lemma

- Suppose that $M=(Q, \Sigma, q_0, A, \delta)$ is an FA accepting L and that **it has n states**
  - If it **accepts** a string x such that **$|x| \geq n$**, then by the time n symbols have been read, M must …

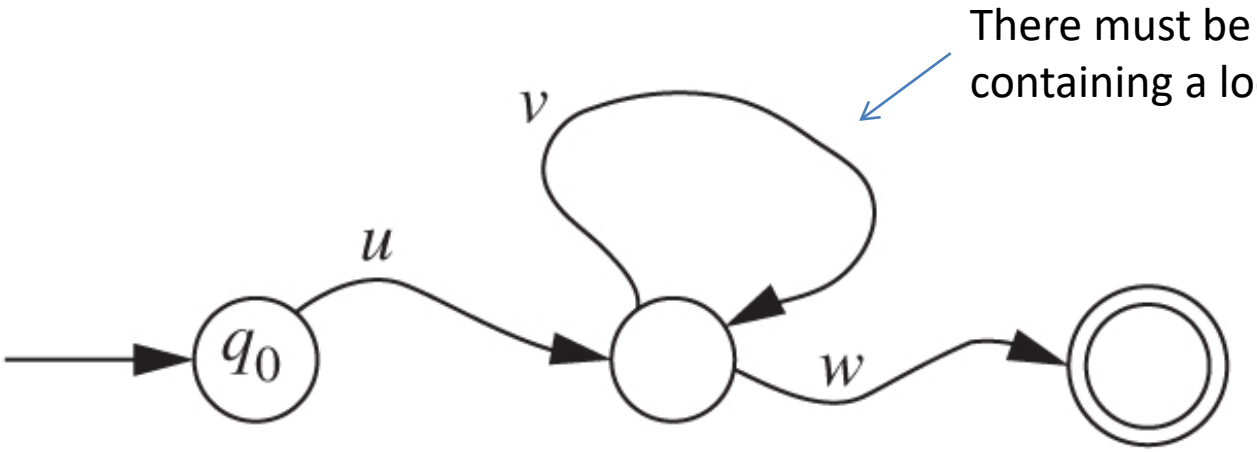
The Pumping Lemma

- Suppose that $M=(Q, \Sigma, q_0, A, \delta)$ is an FA accepting L and that **it has *n* states**
  - If it **accepts** a string x such that $|x| \geq n$, then by the time n symbols have been read, M must have entered some state more than once; i.e., there must be two different prefixes u and uv such that $\delta^*(q_0,u)= \delta^*(q_0,uv)$



The Pumping Lemma (cont'd.)

- This implies that there are many more strings in L, because we can traverse the loop v any number of times (including leaving it out altogether)
- In other words, all of the strings uv^iw for $i \geq 0$ are in L
- This fact is known as the Pumping Lemma for Finite Automata (or for Regular Languages)



The Pumping Lemma

- **Theorem**: Suppose L is a language over Σ

If L is accepted by the FA $M=(Q, \Sigma, q_0, A, \delta)$, and $|Q| = n$, then for every x in L satisfying $|x| \geq n$, there are three strings u, v, and w such that $x = uvw$ and

  - $|uv| \leq n$
  - $|v| > 0$ (i.e. $v \neq \Lambda$)
  - For every $i \geq 0$, the string uv^iw belongs to L

- The way we found n was to take the number of states in an FA accepting L. In many applications we don't need to know this, only that there is such an n

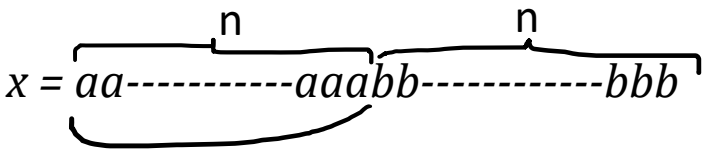
The Pumping Lemma (cont'd.)

- The most common application of the pumping lemma is to show that a language *cannot* be accepted by an FA, because it doesn't have the properties that the pumping lemma says are required for every language that can be.
- The proof is by contradiction. We suppose that the language can be accepted by an FA, and we let $n=|Q|$ be the integer in the pumping lemma
- Then we choose a string x with $|x| \geq n$ to which we can apply the lemma so as to get a contradiction

Example: language that cannot be accepted by an FA, one way to use the pumping lemma

Let L be the language $AnBn = \{a^i b^i \mid i \geq 0\}$; let us prove that it cannot be accepted by an FA

- Suppose, for the sake of contradiction, that L is accepted by an FA; let n be as in the pumping lemma
- Choose $x = a^n b^n$; then $x \in L$ and $|x| \geq n$
- Therefore, by the pumping lemma, there are strings u, v, and w such that $x = uvw$ and the 3 conditions hold
- Because $|uv| \leq n$ and x starts with n a's, all the symbols in u and v are a's; therefore, $v = a^k$ for some $k > 0$

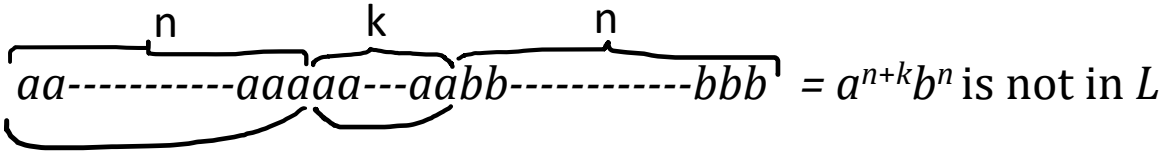


1) uv is here
2) v is also here and consists of a's

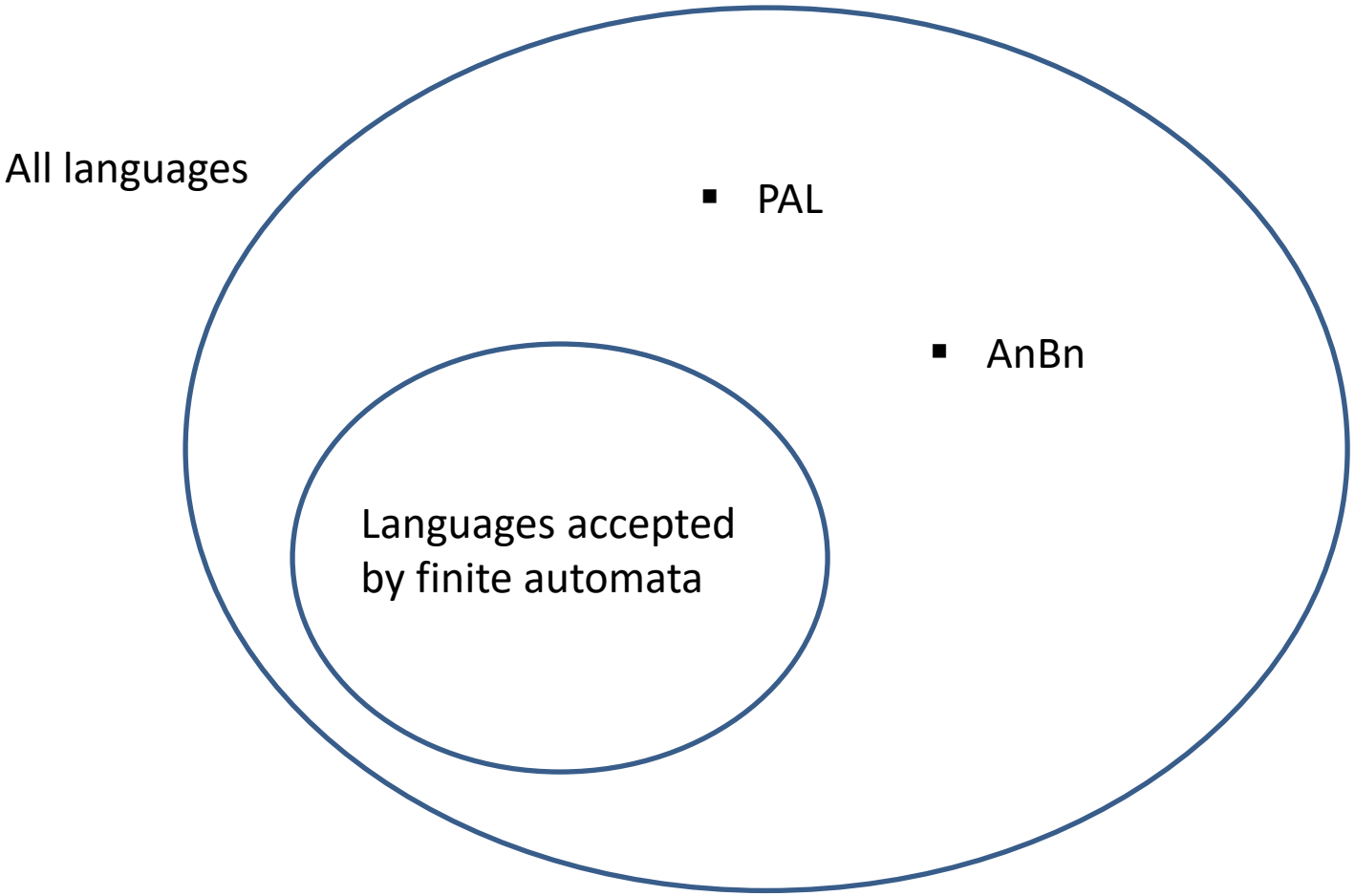
Example: language that cannot be accepted by an FA, one way to use the pumping lemma

Let L be the language $AnBn = \{a^i b^i \mid i \geq 0\}$; let us prove that it cannot be accepted by an FA

- Suppose, for the sake of contradiction, that L is accepted by an FA; let n be as in the pumping lemma
- Choose $x = a^n b^n$; then $x \in L$ and $|x| \geq n$
- Therefore, by the pumping lemma, there are strings u, v, and w such that $x = uvw$ and the 3 conditions hold
- Because $|uv| \leq n$ and x starts with n a's, all the symbols in u and v are a's; therefore, $v = a^k$ for some $k > 0$
- $uvvw \in L$, so $a^{n+k}b^n \in L$. This is our contradiction, and we conclude that L cannot be accepted by an FA



So, what do we know about languages …

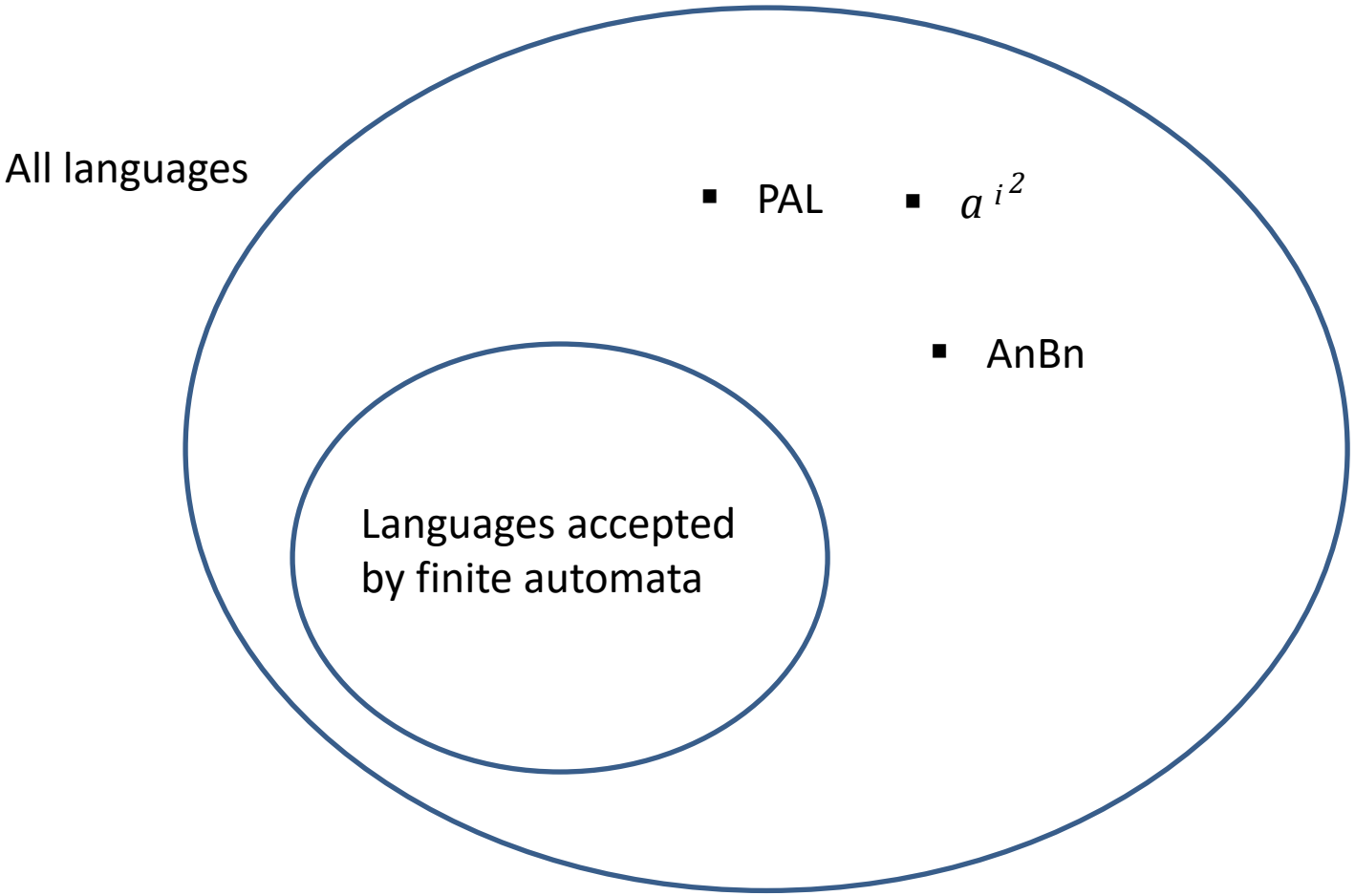


Example 2: language that cannot be accepted by an FA, one way to use the pumping lemma

Let's show $L = \{a^{i^2} \mid i \geq 0\}$ is not accepted by an FA

- Suppose L is accepted by an FA, and let n be the integer in the pumping lemma
- Choose $x = a^{n^2}$ ← (note that the next longer string will be $a^{(n+1)^2}$)
- $x = uvw$, where $0 < |v| \leq n$
- Then $n^2 = |uvw| < |uv^2w| = n^2 + |v| \leq n^2 + n < (n+1)^2$
- This is a contradiction, because $|uv^2w|$ must be i^2 for some integer i (because $uv^2w \in L$), but there is no integer i whose square is strictly between n^2 and $(n+1)^2$

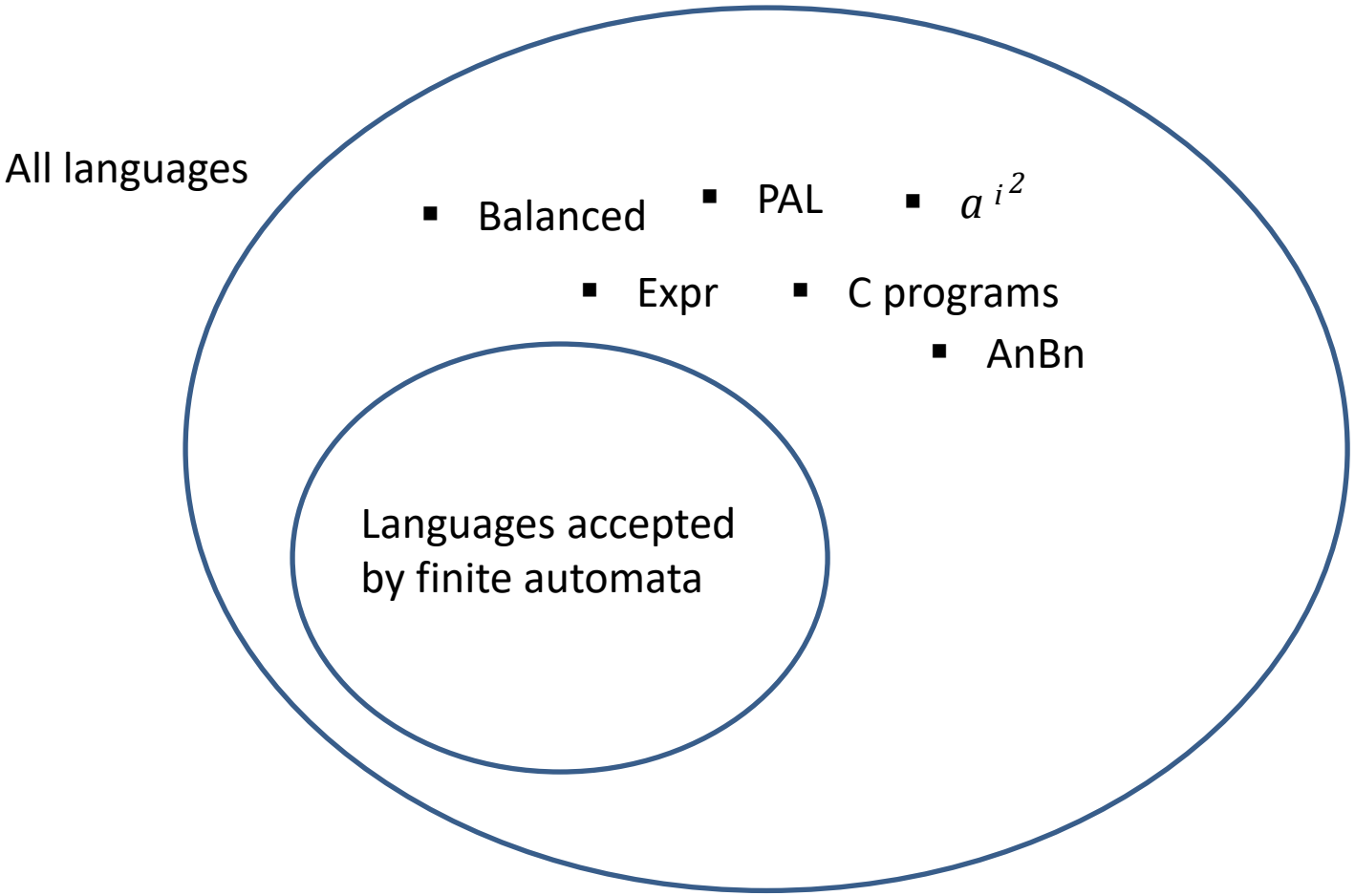
So, what do we know about languages …



The Pumping Lemma (cont'd.)

- There are other languages that are not accepted by any FA, among them:
  - *Balanced*, the set of balanced strings of parentheses
  - *Expr*, the language of simple algebraic expressions
  - The set of legal C programs
- In all three examples, because of the nature of these languages, a proof using the pumping lemma might look a lot like the proof for *AnBn*, our first example
- For example, this string "main(){{{ … }}}" cannot be accepted by an FA (because of $\{^n\}^n$).

So, what do we know about languages …



The Pumping Lemma (cont'd.)

- We can formulate several "decision problems" involving the language L accepted by an FA
  - The membership problem (Given x, is $x \in L(M)$?)
  - Given an n-state FA M, is the language $L(M)$ empty?
    - It follows from the PL that this can be solved by looking at all possible strings of length 0 to n -1; if none of those is accepted, the language is empty, i.e., if we find x that is longer than n, we can always extract a middle part v.
  - Given an n-state FA M, is $L(M)$ infinite?
    - The pumping lemma implies that the language is infinite if and only if at least one of the strings with length from n to $2n$ -1 is accepted

Example: Given two FAs M_1 and M_2, are there any strings that are accepted by neither?

- We know how to construct an FA for the complement of a language L for a given FA, i.e., for $\overline{L} = \Sigma^* - L$.
- Apply this for finding $\overline{M}_1$, and $\overline{M}_2$, accepting $\overline{L(M_1)}$, and $\overline{L_(M_2)}$.
- We can construct an FA accepting $\overline{L(M_1)} \cap \overline{L_(M_2)}$.
- Run the algorithm for determining if this FA accepts any strings.