

# Regular Languages

---

Given alphabet  $\Sigma$ , a *language* is a subset of  $\Sigma^*$ .

Can we write a program to determine whether string  $w$  is in language  $L$ ?

The following languages are called *regular*:

- Basis:  $\emptyset$ ,  $\{\epsilon\}$ , and  $\{a\}$  are regular languages for all  $a \in \Sigma$ .
- Induction: If  $L$  and  $M$  are regular languages, then the following languages are also regular:  $L \cup M$ ,  $LM$  and  $L^*$ .

# Regular Expressions

---

- Basis:  $\emptyset$ ,  $\{\epsilon\}$ , and  $\{a\}$  are regular expressions for all  $a \in \Sigma$ .
- Induction: If  $R$  and  $S$  are regular expressions, then the following expressions are also regular:  $(R)$ ,  $R|S$ ,  $RS$ , and  $R^*$ .

Order of precedence: parentheses, closure, concatenation, alternation.

Every regular expression represents a regular language, and every regular language is represented by a regular expression.

Given regular expression  $R$ ,  $L(R)$  stands for the language represented by  $R$ .

Some properties:

- $R^* = R^*R^* = (R^*)^* = R|R^*$ .
- $R(SR)^* = (RS)^*R$ .
- $(R^*S)^* = \epsilon|(R|S)^*S$ .
- $(RS^*)^* = \epsilon|R(R|S)^*$ .

# Examples

---

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

1.  $a|b$  denotes  $\{a, b\}$
2.  $(a|b)(a|b)$  denotes  $\{aa, ab, ba, bb\}$   
i.e.,  $(a|b)(a|b) = aa|ab|ba|bb$
3.  $a^*$  denotes  $\{\epsilon, a, aa, aaa, \dots\}$
4.  $(a|b)^*$  denotes the set of all strings of  $a$ 's and  $b$ 's  
(including  $\epsilon$ )  
i.e.,  $(a|b)^* = (a^*b^*)^*$
5.  $a|a^*b$  denotes  $\{a, b, ab, aab, aaab, aaaab, \dots\}$

# Finite State Automata

---

Regular languages and regular expressions describe (generate) a certain class of languages.

How do we *recognize* if a string is a member of a particular regular language?

Finite State Automata (FSAs) – a severely restricted model of computation.

- No “stored program” concept – the machine is the computation.
- No auxiliary memory – the automaton fixes memory by its definition.
- Input: a string on a “tape”
- Read head moves over string (left to right).
- Output: “Accept” or “Not Accept”

Advantages of Finite State Automata:

- Well understood
- Clearly defined
- Provides good solutions to useful problems: lexical analysis, pattern matching

# Finite State Automata

---

Finite state automaton operation is completely determined by the input.

Input tape: made of squares, one symbol per square.

CPU is a finite collection of states.

At any instant, finite state automaton is “in” some state.

As symbols are read, finite state automaton may change to another state.

**Kleene's theorem:** A language is regular iff it is accepted by a finite state automaton.

# Deterministic Finite Automata

---

A Deterministic Finite Automaton (DFA) is a quintuple  $M = (Q, \Sigma, \delta, q_0, F)$  where:

- $Q$  is a finite set of *states*.
- $\Sigma$  is an alphabet.
- $q_0 \in Q$  is the *initial state*.
- $F \subseteq Q$  is the set of *final states*.
- $\delta$ , the *transition function*, is from  $Q \times \Sigma$  to  $Q$ .

If  $M$  is in state  $q \in Q$  and the symbol read from input is  $a \in \Sigma$ , then  $\delta(q, a) \in Q$  is the uniquely determined state to which  $M$  passes.

## Example DFA

---

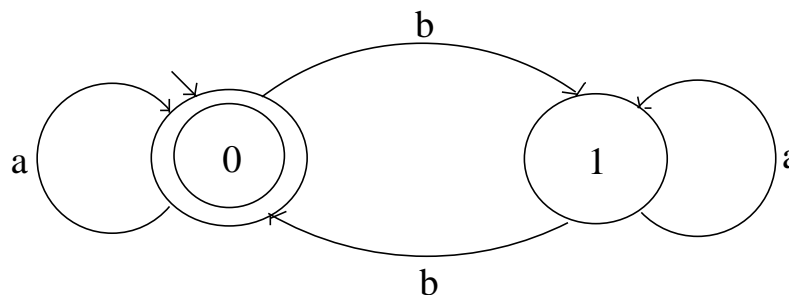
Define  $M = (Q, \Sigma, \delta, q_0, F)$  to be:

- $Q = \{0, 1\}$
- $\Sigma = \{a, b\}$
- $q_0 = 0$
- $F = \{0\}$

and  $\delta$  is the following function:

<i>State</i>	<i>Symbol</i>	$\delta(\textit{State}, \textit{Symbol})$
0	<i>a</i>	0
0	<i>b</i>	1
1	<i>a</i>	1
1	<i>b</i>	0

This can be drawn as:



Consider input *aabba*.

# Configurations

---

Read head only moves right – what has passed has no effect beyond determining current state.

Thus, remaining string and current state completely determine a *configuration*.

A configuration of  $M = (Q, \Sigma, \delta, q_0, F)$  is any element of  $Q \times \Sigma^*$ . Example:  $(0, aabba)$ .

Notation:  $\vdash$ : “yields” is a binary relation on configurations. The relation holds if  $M$  can pass from one configuration to the other.

$(q, w) \vdash_M (q', w')$  iff  $w = aw'$  for  $a \in \Sigma$  and  $\delta(q, a) = q'$ .

$\vdash_M^*$ : transitive closure of  $\vdash_M$ :

$(q, w) \vdash_M^* (q', w')$  means  $(q, w)$  yields  $(q', w')$

after some number, possibly zero, of steps.

## Example

---

Consider input  $aabba$  for the previous DFA  $M$ :

$$\begin{aligned} (0, aabba) &\vdash_M (0, abba) \\ &\vdash_M (0, bba) \\ &\vdash_M (1, ba) \\ &\vdash_M (0, a) \\ &\vdash_M (0, \epsilon) \end{aligned}$$

Therefore,  $(0, aabba) \vdash_M^* (0, \epsilon)$  and so  $aabba$  is accepted by  $M$ .

# DFA Examples

---

1. Design a DFA  $M$  that accepts language  $L(M) = \{w : w \in \{a, b\}^* \text{ and } w \text{ does not contain three consecutive } b\text{'s}\}$ .

2. Design a DFA to accept language  $L = (ab|aba)^*$ .

# Non-deterministic Finite Automata

---

A Non-deterministic Finite Automaton (NFA) is a quintuple  $M = (Q, \Sigma, \delta, q_0, F)$  where:

- $Q$  is a finite set of *states*.
- $\Sigma$  is an alphabet.
- $q_0 \in Q$  is the *initial state*.
- $F \subseteq Q$  is the set of *final states*.
- $\delta$ , the *transition function*, is from  $Q \times (\Sigma \cup \{\epsilon\})$  to  $\wp(Q)$ .

Nondeterminism: Allow a choice of more than one “next state” for a given input symbol.

A Deterministic Finite Automaton (DFA) is a special case of an NFA:

- no transition is labelled with  $\epsilon$
- for each state  $q \in Q$  and input symbol  $a \in \Sigma$ , there is at most one transition labelled  $a$  leaving  $q$

## Example NFA

---

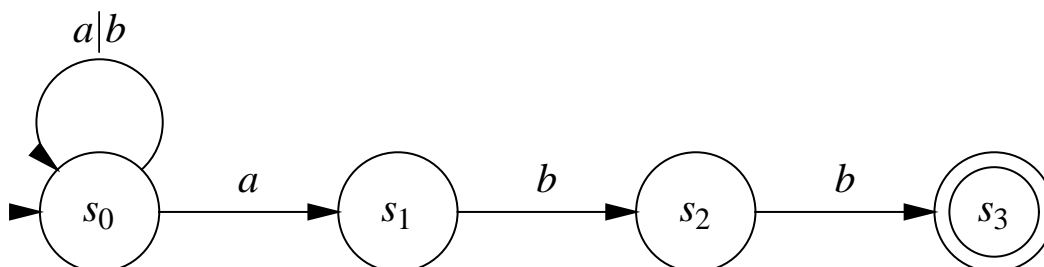
Define  $M = (Q, \Sigma, \delta, q_0, F)$  to be:

- $Q = \{s_0, s_1, s_2, s_3\}$
- $\Sigma = \{a, b\}$
- $q_0 = s_0$
- $F = \{s_3\}$

and  $\delta$  is the following function:

<i>State</i>	<i>Symbol</i>	$\delta(\textit{State}, \textit{Symbol})$
$s_0$	$a$	$s_0$
$s_0$	$b$	$s_0$
$s_0$	$a$	$s_1$
$s_1$	$b$	$s_2$
$s_2$	$b$	$s_3$

This can be drawn as:



# Equivalence of DFA and NFA

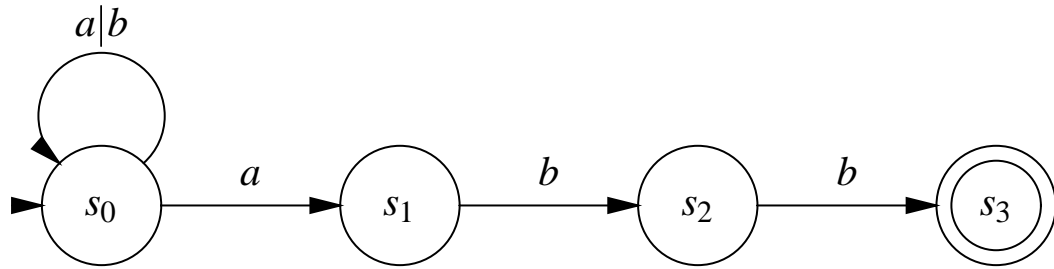
---

Finite automata  $M_1$  and  $M_2$  are *equivalent* iff  $L(M_1) = L(M_2)$

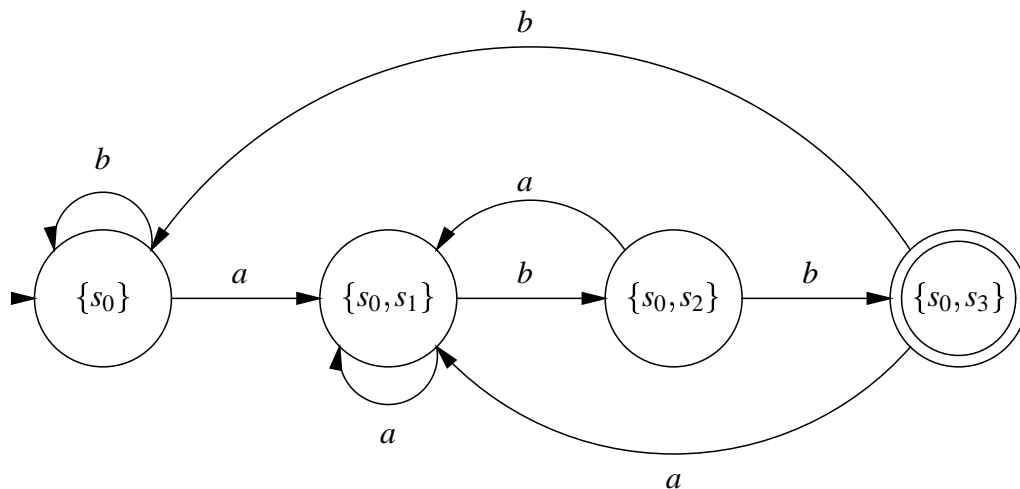
**Theorem:** For each NFA, there is an equivalent DFA.

- DFAs are clearly a subset of NFAs
- Any NFA can be converted into a DFA, by simulating sets of simultaneous states:
  - each DFA state corresponds to a set of NFA states
  - possible exponential blowup in number of states

# Converting NFA to DFA



	$a$	$b$
$\{s_0\}$	$\{s_0, s_1\}$	$\{s_0\}$
$\{s_0, s_1\}$	$\{s_0, s_1\}$	$\{s_0, s_2\}$
$\{s_0, s_2\}$	$\{s_0, s_1\}$	$\{s_0, s_3\}$
$\{s_0, s_3\}$	$\{s_0, s_1\}$	$\{s_0\}$



# Useful Properties

---

The following questions can be answered about finite state automata (FSA):

For FSA  $M$  and string  $w$ , is  $w \in L(M)$ ?

For FSA  $M$ , is  $L(M) = \emptyset$ ?

For FSA  $M$ , is  $L(M) = \Sigma^*$ ?

For FSA  $M_1$  and  $M_2$ , is  $L(M_1) \subseteq L(M_2)$ ?

For FSA  $M_1$  and  $M_2$ , is  $L(M_1) = L(M_2)$ ?

# Grammars

---

A grammar  $G$  is defined as a quadruple  $(S, N, T, P)$ , where:

$S$  is the *start symbol*

$N$  is a set of *non-terminal symbols*

$T$  is a set of *terminal symbols*

$P$  is a set of *productions or rewrite rules*

$(P : N \rightarrow N \cup T)$

The terminal symbols are the basic symbols which belong to the input alphabet.

The non-terminal symbols are defined by the production rules as a concatenation of terminals and non-terminals.

The start symbol is a distinguished non-terminal which is used as the starting point for the definition of the language.

For example, to define the language of sheep:

$$\begin{array}{l} S \rightarrow \text{baa} \\ \quad | \text{baa } S \end{array}$$

# Regular Grammars

---

RE:  $a(a^*|b^*)b$ .

Consider this alternative view:

$$\begin{array}{lcl} S & \rightarrow & aM \\ M & \rightarrow & A \\ M & \rightarrow & B \\ A & \rightarrow & b \\ A & \rightarrow & aA \\ B & \rightarrow & b \\ B & \rightarrow & bB \end{array}$$

To generate the string  $aaab$ :

$$S \rightarrow aM \rightarrow aA \rightarrow aaA \rightarrow aaaA \rightarrow aaab$$

# Regular Grammars

---

Can we place a restriction on the *form* of a grammar to ensure that it describes a regular language?

**Theorem:** For any RE  $r$ , there is a grammar  $g$  such that  $L(r) = L(g)$ .

The grammars that generate regular sets are called *regular grammars*

**Definition:** In a regular grammar, all productions have one of two forms:

1.  $A \rightarrow aB$

2.  $A \rightarrow a$

where  $A$ ,  $B$  are any non-terminals and  $a$  is any terminal symbol.

These are also called *type 3* grammars (Chomsky)

# Regular Grammars

---

Every regular grammar can be converted to a finite-state automaton, and vice-versa.

To create a FSA corresponding to a regular grammar:

- the input symbols of the FSA are the terminal symbols of the grammar
- the states in the FSA are the non-terminals of the grammar (plus one extra state,  $F$  say) such that:
  - the start state is the state corresponding to the start symbol
  - the (only) final state is  $F$
- the transitions of the FSA correspond to the production rules of the grammar as follows:
  - if the rule is of the form  $A \rightarrow aB$ , create a new transition of the form  $\delta(A, a) = B$
  - if the rule is of the form  $A \rightarrow a$ , create a new transition of the form  $\delta(A, a) = F$

# Regular Grammars

---

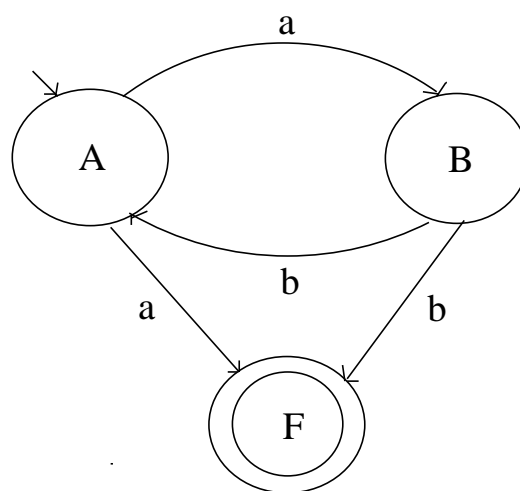
For example, consider the following regular grammar:

$$\begin{array}{lcl}
 A & \rightarrow & a \\
 & & | \quad aB \\
 B & \rightarrow & b \\
 & & | \quad bA
 \end{array}$$

Using the rules above, we can construct the following corresponding FSA:

<i>State</i>	<i>Symbol</i>	$\delta(\textit{State}, \textit{Symbol})$
<i>A</i>	<i>a</i>	<i>F</i>
<i>A</i>	<i>a</i>	<i>B</i>
<i>B</i>	<i>b</i>	<i>F</i>
<i>B</i>	<i>b</i>	<i>A</i>

This can be drawn as:



## Limits of regular languages

---

Not all languages are regular

One cannot construct DFAs to recognize these languages:

- $L = \{p^k q^k\}$
- $L = \{w \in \{a, b\}^* \mid w \text{ has an equal number of 'a's and 'b's}\}$
- $L = \{w c w^r \mid w \in \Sigma^*\}$ , where  $w^r$  is  $w$  reversed

*Note: neither of these is a regular expression!*  
(DFAs cannot count!)

But, this is a little subtle. One can construct DFAs for:

- alternating 0's and 1's  
 $(\epsilon|1)(01)^*(\epsilon|0)$
- sets of pairs of 0's and 1's  
 $(01|10)^+$

# Pumping Lemma

---

The pumping lemma for regular languages can be stated as follows:

Let  $L$  be an infinite regular language. Then there are strings  $x$ ,  $y$  and  $z$  such that  $y \neq \epsilon$  and  $xy^kz \in L$  for each  $k \geq 0$ .

## Proof:

- $L$  is accepted by some DFA  $M$  (with a finite number of states).
- Since  $L$  is infinite and  $M$  has a finite number of states, there must be a cycle in  $M$ .
- This cycle corresponds to  $y$ , and can be repeated an arbitrary number of times.

For example, the language  $L = \{p^k q^k\}$  cannot be rewritten as  $xy^kz$ , so it cannot be regular.