

Deterministic Finite Automata

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

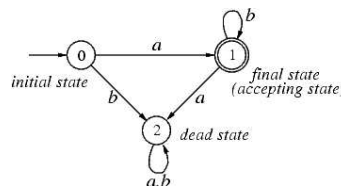


Deterministic Finite Automata

Introduction

Consider the language matched by the regular expression ab^* . This language consists of all strings that start with a and having zero or more b 's following it.

Suppose that we wanted a "machine" that could examine a string and determine if it was in this language or not. The following figure is a **transition diagram** of such a machine:



Try walking through this with the strings:

- $w = abbb$
- $w = a$
- $w = abab$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

Introduction

The rules for tracing a string through a transition diagram are pretty straightforward.

We always start at the **initial** or **start state** and transition from one state to another (including possibly staying at the same state) based on the symbols in the input string.

As long as we end up in a **final** or **accepting state** (denoted with two concentric circles) then we say that the string is accepted by the machine. If we are not in an accepting state when we run out of symbols in the input string then the string is not accepted the machine.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

"Loose" Definition

A **deterministic finite automaton** (DFA) is a collection of three things:

1. A finite set of states, one of which is designated as the initial state, called the **start state**, and some (maybe none) of which are designated as **final states**.
2. An **alphabet** Σ of all possible input symbols.
3. A finite set of **transitions** that tell for each state and for each symbol of the input alphabet which state to go to next.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

Formal Definition

A **deterministic finite automaton** is a 5-tuple $M = (Q, \Sigma, \delta, s, F)$ consisting of:

1. A finite collection of **states** Q .
2. An input alphabet Σ .
3. A **transition function** $\delta: Q \times \Sigma \Rightarrow Q$.
4. An **initial state** $s \in Q$.
5. A collection of **final** or **accepting states** $F \subseteq Q$.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

Transition Function

There are several ways to describe the transition function. One is to use a transition diagram, as we have already seen. Note that since this *is a function*, a transition must be defined for every symbol-state pair.

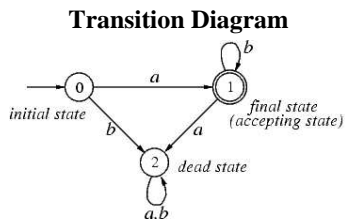
Thus, in a transition diagram there must be a labeled edge for every symbol in Σ initiating at every state.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

Transition Function



Transition Table

q	a	b
q_0	q_1	q_2
q_1	q_2	q_1
q_2	q_2	q_2

Notice how the transition table has an entry for every possible combination of state and input symbol. The fact that δ is a function is what makes this theoretical machine deterministic.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

Input Tape

We've seen several ways to describe deterministic finite automata, but we need something more for these DFAs to do anything.

The basic operation of a DFA is to process an input string, usually called the **input tape**. The input tape contains symbols from Σ and a **blank** symbol not in Σ . A **tape head** is initially located at the start of the input tape.

Each step of processing with the DFA involves examining the current state and the symbol under the tape head, making the transition indicated by this pair, and advancing the tape head to the next symbol on the tape. When the tape head is on a blank the machine halts.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

The Language of a DFA

If the machine halts in an accepting or final state we say that the machine **accepts** the string on the input tape, otherwise we say that machine **rejects** the string.

The **language of a DFA** is the set of strings that it accepts.

Thus, DFAs can be used to define a language (i.e., they can represent a language), much like a regular expression can.

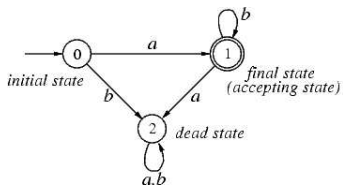
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

Derivation

We can trace how a particular DFA works. Consider again our DFA that accepts strings matching ab^* and trace it on the string $abbab$:



$(q_0, abbab) \vdash (q_1, bbab)$
 $\vdash (q_1, bab)$
 $\vdash (q_1, ab)$
 $\vdash (q_2, b)$
 $\vdash (q_2, \Lambda)$

The final ordered pair (q_2, Λ) indicates that the input has been consumed and the tape head is over a blank. Since the state q_2 is not an accepting state we know that the string $abbab$ is rejected by this DFA.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

Configuration and Derivation

Assume that M is a DFA. If we know the state that M is currently in and the unprocessed portion of the input string, we have all we need to know to determine how M will behave from that point on. This information is called the **configuration of the DFA** and is denoted with the ordered pair (q,w) .

We will use the notation

$$(q,w) \vdash_M (q', w')$$

to indicate that when M is in the configuration (q,w) one step of the DFA will leave it in the new configuration (q',w') . We say that (q,w) **derives** or **yields** the configuration (q',w') . When it's clear from the context which DFA we are discussing we can drop the name of the DFA from the derivation symbol and just use \vdash .

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

Derivation

If we want to indicate that zero or more steps of a derivation we will use the symbol \vdash^* .

In our last example we have that $(q_0, abbab) \vdash^* (q_2, \Lambda)$. Several other examples are

$(q_0, abbbb) \vdash^* (q_1, \Lambda),$
 $(q_0, bbba) \vdash^* (q_2, \Lambda),$
 $(q_0, abab) \vdash^* (q_2, \Lambda),$

only the first of which results in an accepting configuration.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

DFAs from Regular Expressions

Assume that $\Sigma = \{a, b\}$.

Construct a DFA that accepts the same strings as the regular expression $(ab)^*$.

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Deterministic Finite Automata

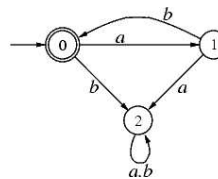
[prev](#) | [slides](#) | [next](#)

DFAs from Regular Expressions

Assume that $\Sigma = \{a, b\}$.

Construct a DFA that accepts the same strings as the regular expression $(ab)^*$.

Solution:



[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Deterministic Finite Automata

[prev](#) | [slides](#) | [next](#)

DFAs from Regular Expressions

Assume that $\Sigma = \{a, b\}$.

Construct a DFA that accepts the same strings as the regular expression $ab^*(ab)^*$.

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Deterministic Finite Automata

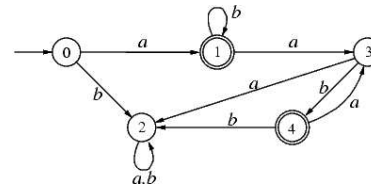
[prev](#) | [slides](#) | [next](#)

DFAs from Regular Expressions

Assume that $\Sigma = \{a, b\}$.

Construct a DFA that accepts the same strings as the regular expression $ab^*(ab)^*$.

Solution:



[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)