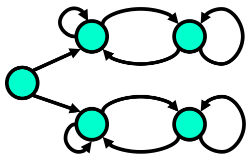
Great Theoretical Ideas in CS

V. Adamchik

CS 15-251

Carnegie Mellon University

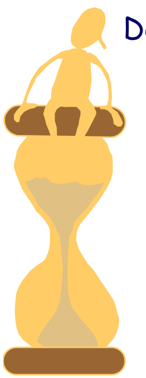
Finite Automata



Outline

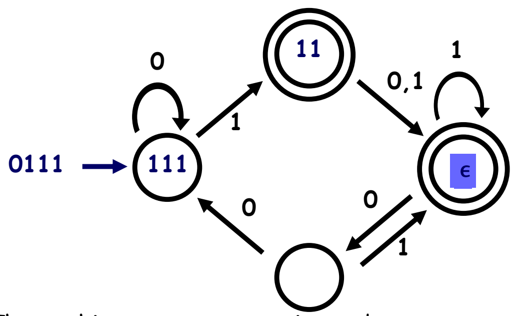
DFA
Regular Languages
0^n1^n is not regular
Union Theorem
Kleene's Theorem
NFA
Application: KMP

Deterministic Finite Automaton



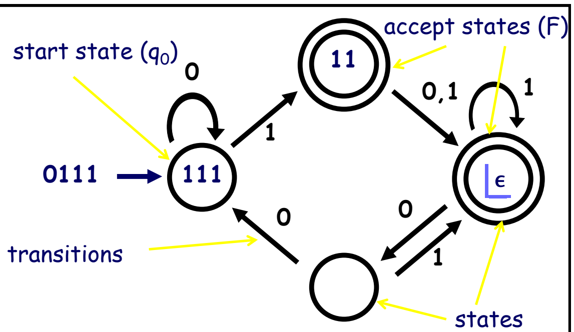
A machine so simple that you can understand it in just one minute

☺



The machine processes a string and accepts it if the process ends in a double circle

The unique string of length 0 will be denoted by ε and will be called the empty or null string



The machine accepts a string if the process ends in an accept state (double circle)

Anatomy of a Deterministic Finite Automaton

The singular of automata is automaton.

The alphabet Σ of a finite automaton is the set where the symbols come from, for example {0,1}

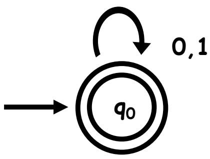
The language L(M) of a finite automaton is the set of strings that it accepts

L(M) = {x∈Σ: M accepts x}

It's also called the

"language decided/accepted by M".

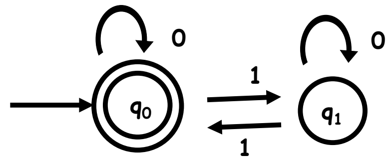
The Language L(M) of Machine M



L(M) = All strings of 0s and 1s

The language of a finite automaton is the set of strings that it accepts

The Language L(M) of Machine M



What language does this DFA decide/accept?

L(M) = { w | w has an even number of 1s}

Formal definition of DFAs

A finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

Q is the finite set of states

Σ is the alphabet

$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept states

L(M) = the language of machine M
= set of all strings machine M accepts

$M = (Q, \Sigma, \delta, q_0, F)$
where

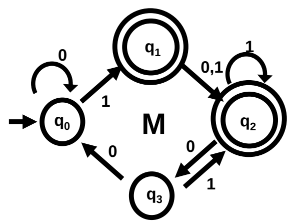
$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0,1\}$

$q_0 \in Q$ is start state

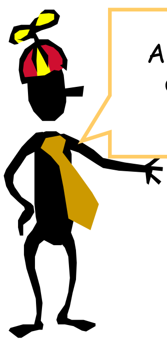
$F = \{q_1, q_2\} \subseteq Q$ accept states

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

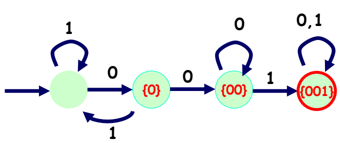


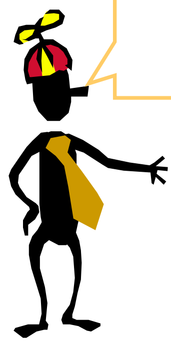
δ	0	1
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_3	q_2
q_3	q_0	q_2

EXAMPLE

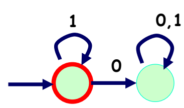


An automaton that accepts all and only those strings that contain 001

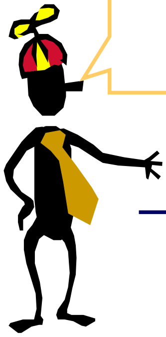




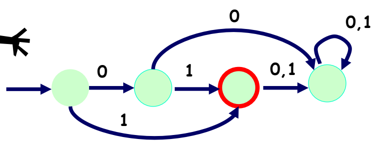
Determine the language recognized by



L(M)={1,11,111, …}



Determine the language decided by



L(M)={1, 01}

Membership problem

Determine whether some word belongs to the language.

Regular Languages

A language over Σ is a set of strings over Σ

A language L ⊆ Σ is regular if it is recognized by a deterministic finite automaton

A language L ⊆ Σ is regular if there is a DFA which decides it.

L = { w | w contains 001} is regular

L = { w | w has an even number of 1s} is regular

DFA Membership problem

Determine whether some word belongs to the language.

Theorem: The DFA Membership Problem is solvable in linear time.

Let M = (Q, Σ, δ, q_0, F) and w = $w_1 \ldots w_m$.
Algorithm for DFA M:

p := q_0;
for i := 1 to m do p := $\delta(p, w_i)$;
if p∈F then return Yes else return No.

Are all languages regular?

Theorem: Any finite language is regular

Theorem: Any finite language is regular

Claim 1: Let w be a string over an alphabet. Then {w} is a regular language.

Proof: By induction on the number of characters. If {a} and {b} are regular then {ab} is regular

Claim 2: A language consisting of n strings is regular

Proof: By induction on the number of strings. If {a} then L∪{a} is regular

Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Notation:

If $a \in \Sigma$ is a symbol and $n \in N$ then a^n denotes the string aaa···a (n times).

E.g., a^3 means aaa, a^5 means aaaaa, a^1 means a, a^0 means ϵ, etc.

Thus $L = \{\epsilon, 01, 0011, 000111, 00001111, \ldots\}$.

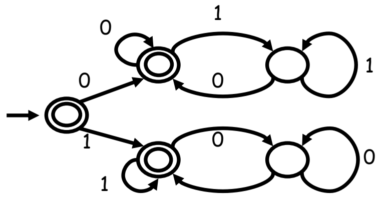
Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Wrong Intuition:

For a DFA to decide L, it *seems* like it needs to "remember" how many 0's it sees at the beginning of the string, so that it can "check" there are equally many 1's.

But a DFA has only finitely many states — shouldn't be able to handle arbitrary n.

L = strings where the number of occurrences of 01 is equal to the number of occurrences of 10



M accepts only the strings with an equal number of 01's and 10's!
For example, 010110

How to prove a language is NOT regular…

Assume for contradiction there is a DFA M with L(M) = L.

Argue (usually by Pigeonhole) there are two strings x and y which reach the same state in M.

Show there is a string z such that $xz \in L$ but $yz \notin L$. Contradiction, since M accepts either both (or neither.)

Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Full proof:

Suppose M is a DFA deciding L with, say, k states.

Let r_i be the state M reaches after processing 0^i.

By Pigeonhole, there is a repeat among $r_0, r_1, r_2, \ldots, r_k$. So say that $r_s = r_t$ for some $s \neq t$.

Since $0^s1^s \in L$, starting from r_s and processing 1^s causes M to reach an accepting state.

Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Full proof:

So on input $0^s1^s \in L$, M will reach an accepting state.

Consider input $0^t1^s \notin L$, $s \neq t$.

M will process 0^t, reach state $r_t = r_s$

then M will process 1^s, and reach an accepting state.

Contradiction!

Regular Languages

Definition: A language $L \subseteq \Sigma$ is regular if there is a DFA which decides it.

Questions:

1. Are all languages regular?
2. Are there other ways to tell if L is regular?

Equivalence of two DFAs

Definition: Two DFAs M_1 and M_2 over the same alphabet are equivalent if they accept the same language: $L(M_1) = L(M_2)$.

Given a few equivalent machines, we are naturally interested in the smallest one with the least number of states.

Union Theorem

Given two languages, L_1 and L_2, define the union of L_1 and L_2 as

$L_1 \cup L_2 = \{ w \mid w \in L_1 \text{ or } w \in L_2 \}$

Theorem: The union of two regular languages is also a regular language.

Theorem: The union of two regular languages is also a regular language

Proof (Sketch): Let

$M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ be finite automaton for L_1

and

$M_2 = (Q_2, \Sigma, \delta_2, q_0, F_2)$ be finite automaton for L_2

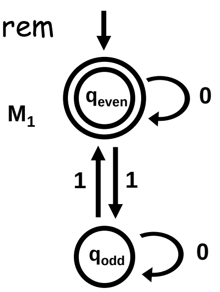
We want to construct a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes $L = L_1 \cup L_2$

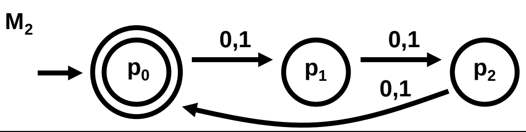
Idea: Run both M_1 and M_2 at the same time.

Union Theorem

L_1 = strings with even # of 1's

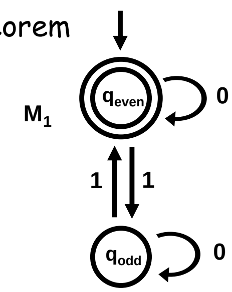
L_2 = strings x with |x| div. by 3

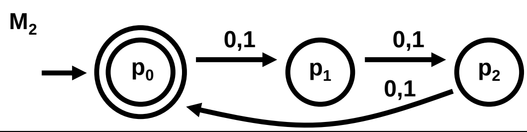


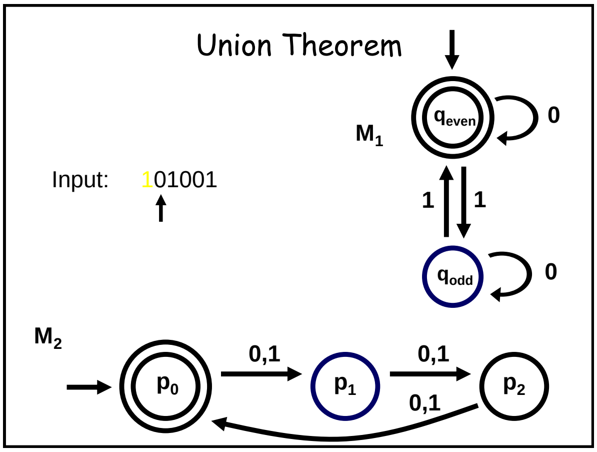


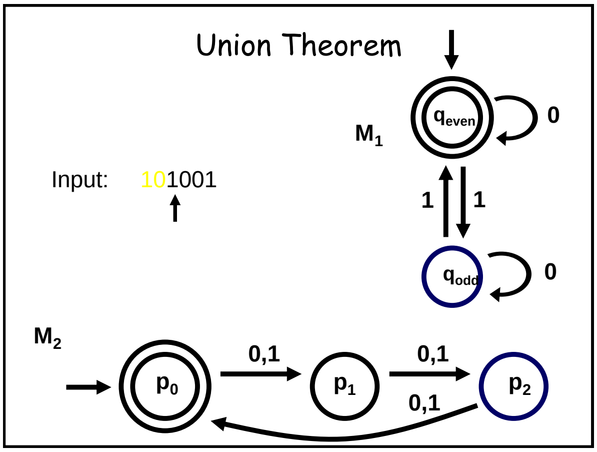
Union Theorem

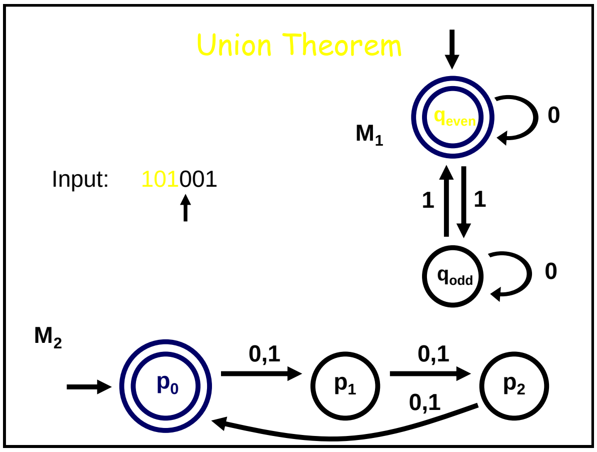
Input: 101001

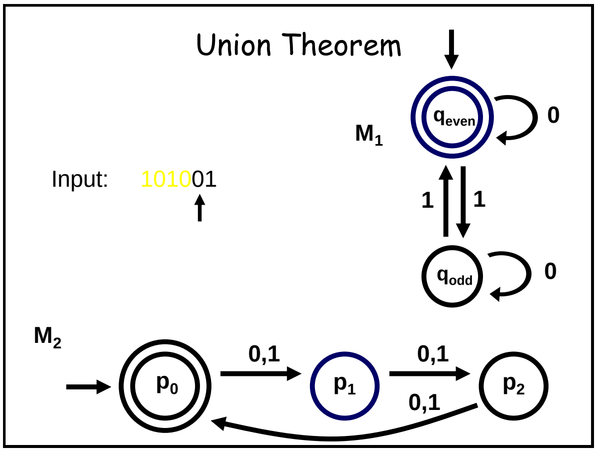


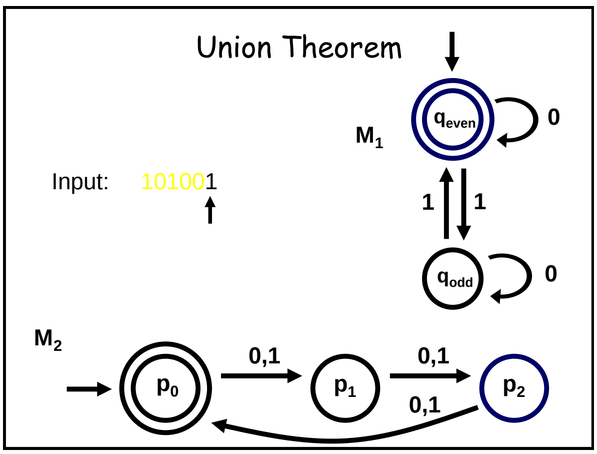


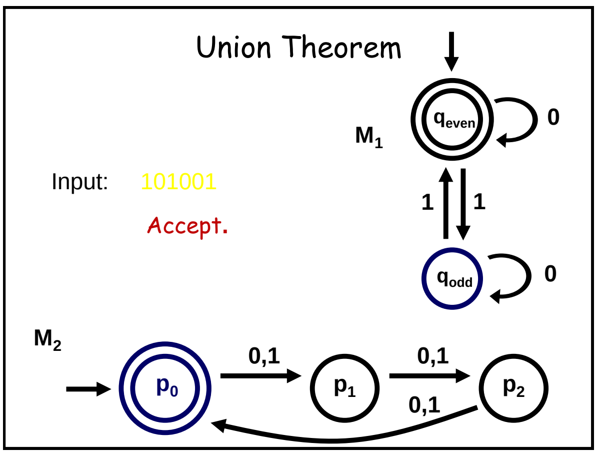

Union Theorem
M1
q_even
0
1
1
q_odd
0
Input: 101001
M2
p0
0,1
p1
0,1
p2
0,1


Union Theorem
M1
q_even
0
1
1
q_odd
0
Input: 101001
M2
p0
0,1
p1
0,1
p2
0,1


Union Theorem
M1
q_even
0
1
1
q_odd
0
Input: 101001
M2
p0
0,1
p1
0,1
p2
0,1

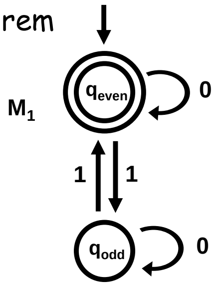

Union Theorem
M1
q_even
0
1
1
q_odd
0
Input: 101001
M2
p0
0,1
p1
0,1
p2
0,1

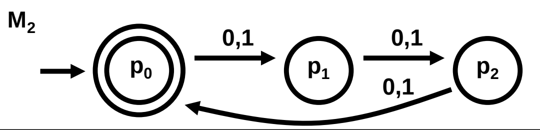

Union Theorem
M1
q_even
0
1
1
q_odd
0
Input: 101001
M2
p0
0,1
p1
0,1
p2
0,1


Union Theorem
M1
q_even
0
1
1
q_odd
0
Input: 101001
Accept.
M2
p0
0,1
p1
0,1
p2
0,1

Union Theorem

Make a DFA keeping track of both at once.



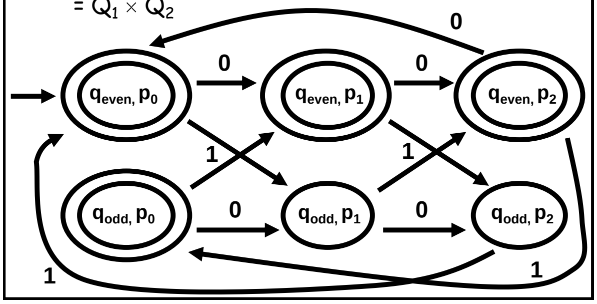


Union Theorem

Q = pairs of states, one from M_1 and one from M_2

= $\{ (q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2 \}$

= $Q_1 \times Q_2$



The Regular Operations

Union: $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$

Intersection: $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$

Negation: $\neg A = \{ w \mid w \notin A \}$

Reverse: $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \}$

Concatenation: $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

Star: $A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \}$

The Kleene closure: A*

Star: $A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \}$

From the definition of the concatenation, we definite A^n, n =0, 1, 2, … recursively

$A^0 = \{\varepsilon\}$

$A^{n+1} = A^n A$

A* is a set consisting of concatenations of arbitrary many strings from A.

$$A^* = \bigcup_{k=0}^{\infty} A^k$$

The Kleene closure: A*

What is A* of A={0,1}?

All binary strings

What is A* of A={11}?

All binary strings of an even number of 1s

Regular Languages Are Closed Under The Regular Operations

An axiomatic system for regular languages

Vocabulary: Languages over alphabet Σ

Axioms: $\emptyset$, {a} for each $a \in \Sigma$

Deduction rules:

Given L_1, L_2, can obtain $L_1 \cup L_2$

Given L_1, L_2, can obtain $L_1 \cdot L_2$

Given L, can obtain L^*

The Kleene Theorem (1956)

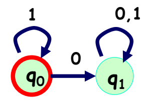
Every regular language over Σ can be constructed from $\emptyset$ and $\{a\}$, $a \in \Sigma$, using only the operations union, concatenation and the Kleene star.

Reverse

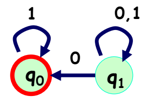
Reverse: $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \}$

How to construct a DFA for the reversal of a language?

The direction in which we read a string should be irrelevant.

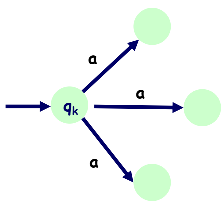


If we flip transitions around we might not get a DFA.



Nondeterministic Finite Automaton

There is another type machine in which there may be several possible next states. Such machines called nondeterministic.



Allows transitions from q_k on the same symbol to many states

Nondeterministic finite automaton (NFA)

Nondeterminism can arise from two different sources:

-Transition nondeterminism

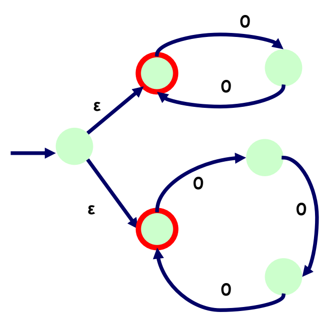
-Initial state nondeterminism

Nondeterministic finite automaton (NFA)

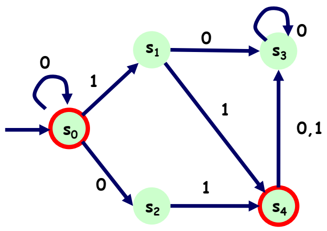
An NFA is defined using the same notations $M = (Q, \Sigma, \delta, I, F)$ as DFA except the initial states I and the transition function δ assigns a set of states to each pair $Q \times \Sigma$ of state and input.

Note, every DFA is automatically also NFA.

NFA for $\{0^k \mid k \text{ is a multiple of 2 or 3}\}$

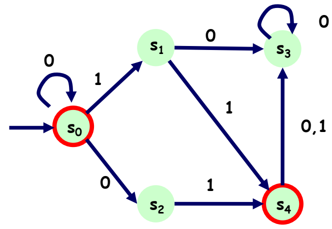


Find the language recognized by this NFA



$L = \{0^n, 0^n01, 0^n11 \mid n = 0, 1, 2\ldots\}$

What does it mean that for an NFA to recognize a string?



Since each input symbol x_j (for j>1) takes the previous state to a set of states, we shall use a union of these states.

What does it mean that for a NFA to recognize a string?

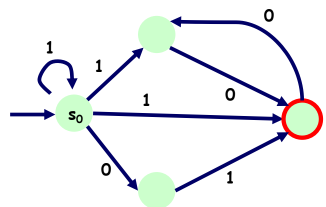
Here we are going formally define this.

For a state q and string w, $\delta^*(q, w)$ is the set of states that the NFA can reach when it reads the string w starting at the state q.

Thus for NFA= $(Q, \Sigma, \delta, q_0, F)$, the function
$\delta^*: Q \times \Sigma \to 2^Q$

is defined by $\delta^*(q, y\, x_k) = \cup_{p \in \delta^*(q,y)} \delta(p, x_k)$

Find the language recognized by this NFA



$L = 1^* (01, 1, 10) (00)^*$

Nondeterministic finite automaton

Theorem.
If the language L is recognized by an NFA, then L is also recognized by a DFA.

In other words,
if we ask if there is a NFA that is not equivalent to any DFA. The answer is No.

Nondeterministic finite automaton

Theorem (Rabin, Scott 1959).

For every NFA there is an equivalent DFA.

For this they won the Turing Award.

Rabin

Scott

CMU prof. emeritus

NFA vs. DFA

Advantages.
Easier to construct and manipulate.
Sometimes exponentially smaller.
Sometimes algorithms much easier.

Drawbacks
Acceptance testing slower.
Sometimes algorithms more complicated.

Pattern Matching

Input: Text T of length k, string/pattern P of length n

Problem: Does pattern P appear inside text T?

Naïve method:

$\boxed{a_1, a_2, a_3,} a_4, a_5, \ldots, a_n$

Cost: Roughly O(n k) comparisons

may occur in images and DNA sequences
unlikely in English text

Pattern Matching

Input: Text T, length n. Pattern P, length k.

Output: Does P occur in T?

Automata solution:

The language P is regular!

There is some DFA M_P which decides it.

Once you build M_P, feed in T: takes time O(n).

Build DFA from pattern

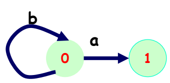
The alphabet is {a, b}.
The pattern is a a b a a a b b.

To create a DFA we consider all prefixes
ε, a, aa, aab, aaba, aabaa, aabaaa, aabaaab, aabaaabb

These prefixes are states. The initial state is ε. The pattern is the accepting state.

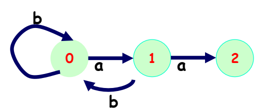
DFA Construction

a a b a a a b b
↑



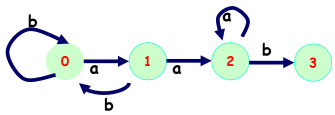
DFA Construction

a a b a a a b b
↑



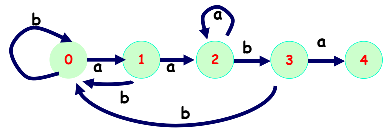
DFA Construction

a a b a a a b b



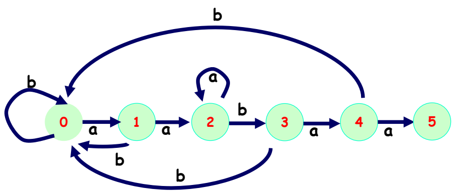
DFA Construction

a a b a a a b b



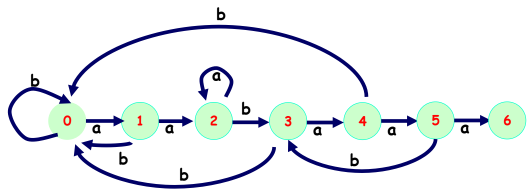
DFA Construction

a a b a a a b b



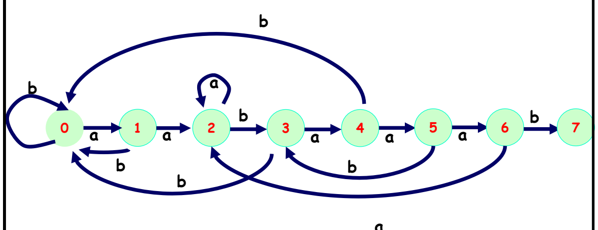
DFA Construction

a a b a a a b b



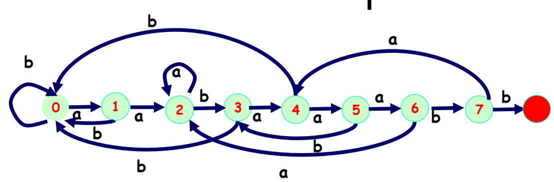
DFA Construction

a a b a a a b b



DFA Construction

a a b a a a b b

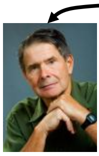


The Knuth-Morris-Pratt Algorithm (1976)

1970 Cook published a paper about a possibility of existence of a linear time algorithm

Knuth and Pratt developed an algorithm

Morris discovered the same algorithm

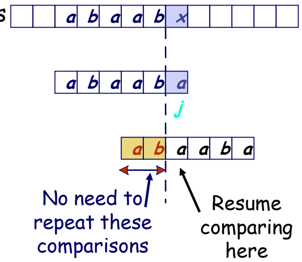




Pittsburgh native, CMU professor.

The KMP Algorithm - Motivation

Algorithm compares the pattern to the text in left-to-right, but shifts the pattern more intelligently than the brute-force algorithm.

When a mismatch occurs, we compute the length of the longest prefix of P that is a proper suffix of P.



Here's What You Need to Know…

- Languages
- DFAs
- The regular operations
- 0^n1^n is not regular
- Union Theorem
- Kleene's Theorem
- NFAs
- Application: KMP