

CS 373: Theory of Computation

Manoj Prabhakaran Mahesh Viswanathan

University of Illinois, Urbana-Champaign

Fall 2008

Decision Problems

Decision Problems

Given input, decide “yes” or “no”

- **Examples:** Is x an even number? Is x prime? Is there a path from s to t in graph G ?
- i.e., Compute a boolean function of input

Decision Problems

Decision Problems

Given input, decide “yes” or “no”

- **Examples:** Is x an even number? Is x prime? Is there a path from s to t in graph G ?
- i.e., Compute a boolean function of input

General Computational Problem

In contrast, typically a problem requires computing some non-boolean function, or carrying out interactive/reactive computation in a distributed environment

- **Examples:** Find the factors of x . Find the balance in account number x .

Decision Problems

Decision Problems

Given input, decide “yes” or “no”

- **Examples:** Is x an even number? Is x prime? Is there a path from s to t in graph G ?
- i.e., Compute a boolean function of input

General Computational Problem

In contrast, typically a problem requires computing some non-boolean function, or carrying out interactive/reactive computation in a distributed environment

- **Examples:** Find the factors of x . Find the balance in account number x .

- In this course, we will study decision problems because aspects of computability are captured by this special class of problems

What Does a Computation Look Like?

- Some code (a.k.a **control**): the same for all instances

What Does a Computation Look Like?

- Some code (a.k.a **control**): the same for all instances
- The input (a.k.a problem instance)

What Does a Computation Look Like?

- Some code (a.k.a **control**): the same for all instances
- The input (a.k.a problem instance)
- As the program starts executing, some memory (a.k.a **state**)

What Does a Computation Look Like?

- Some code (a.k.a **control**): the same for all instances
- The input (a.k.a problem instance)
- As the program starts executing, some memory (a.k.a **state**)
 - Includes the values of variables (and the “program counter”)

What Does a Computation Look Like?

- Some code (a.k.a **control**): the same for all instances
- The input (a.k.a problem instance)
- As the program starts executing, some memory (a.k.a **state**)
 - Includes the values of variables (and the “program counter”)
 - State evolves throughout the computation

What Does a Computation Look Like?

- Some code (a.k.a **control**): the same for all instances
- The input (a.k.a problem instance)
- As the program starts executing, some memory (a.k.a **state**)
 - Includes the values of variables (and the “program counter”)
 - State evolves throughout the computation
 - Often, takes more memory for larger problem instances

What Does a Computation Look Like?

- Some code (a.k.a **control**): the same for all instances
- The input (a.k.a problem instance)
- As the program starts executing, some memory (a.k.a **state**)
 - Includes the values of variables (and the “program counter”)
 - State evolves throughout the computation
 - Often, takes more memory for larger problem instances
- But some programs do not need larger state for larger instances!

Finite State Computation

- **Finite state:** A fixed upper bound on the size of the state, independent of the size of the input

Finite State Computation

- **Finite state:** A fixed upper bound on the size of the state, independent of the size of the input
 - If t -bit state, at most 2^t possible states

Finite State Computation

- **Finite state:** A fixed upper bound on the size of the state, independent of the size of the input
 - If t -bit state, at most 2^t possible states
- Not enough memory to hold the entire input

Finite State Computation

- **Finite state**: A fixed upper bound on the size of the state, independent of the size of the input
 - If t -bit state, at most 2^t possible states
- Not enough memory to hold the entire input
 - “Streaming input”: automaton runs (i.e., changes state) on seeing each bit of input

An Automatic Door

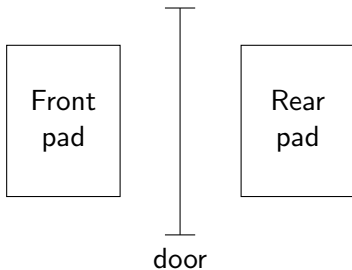


Figure: Top view of Door

An Automatic Door

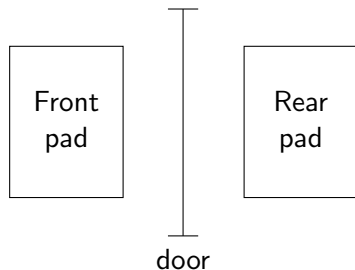


Figure: Top view of Door

- **Input:** A stream of events `<front>`, `<rear>`, `<both>`, `<neither>` ...

An Automatic Door

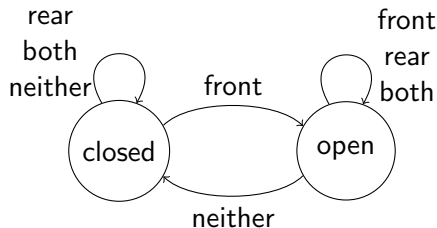
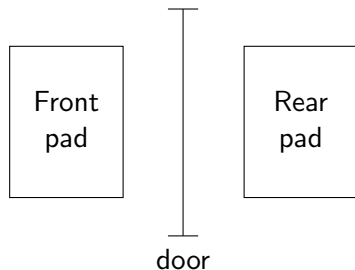


Figure: Top view of Door

Figure: State diagram of controller

- **Input:** A stream of events `<front>`, `<rear>`, `<both>`, `<neither>` ...

An Automatic Door

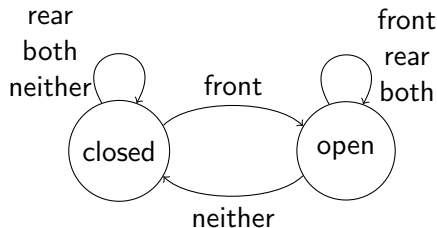
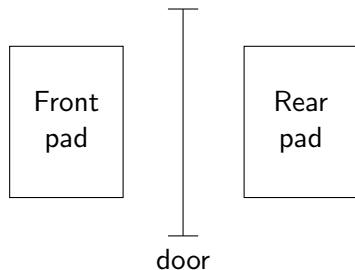


Figure: Top view of Door

Figure: State diagram of controller

- **Input:** A stream of events `<front>`, `<rear>`, `<both>`, `<neither>` ...
- Controller has a single bit of state.

Finite Automata

Details

Automaton

A finite automaton has:

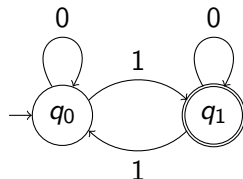


Figure: Transition Diagram of automaton

Finite Automata

Details

Automaton

A finite automaton has: Finite set of states, with **start/initial** and **accepting/final** states;

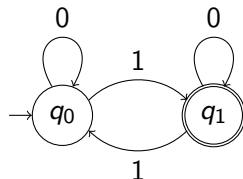


Figure: Transition Diagram of automaton

Finite Automata

Details

Automaton

A finite automaton has: Finite set of states, with **start/initial** and **accepting/final** states; **Transitions** from one state to another on reading a symbol from the input.

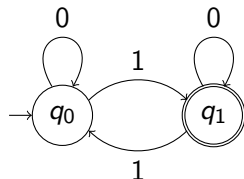


Figure: Transition Diagram of automaton

Finite Automata

Details

Automaton

A finite automaton has: Finite set of states, with **start/initial** and **accepting/final** states; **Transitions** from one state to another on reading a symbol from the input.

Computation

Start at the initial state; in each step, read the next symbol of the input, take the transition (edge) labeled by that symbol to a new state.

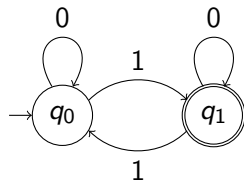


Figure: Transition Diagram of automaton

Finite Automata

Details

Automaton

A finite automaton has: Finite set of states, with **start/initial** and **accepting/final** states; **Transitions** from one state to another on reading a symbol from the input.

Computation

Start at the initial state; in each step, read the next symbol of the input, take the transition (edge) labeled by that symbol to a new state. **Acceptance/Rejection**: If after reading the input w , the machine is in a final state then w is **accepted**; otherwise w is **rejected**.

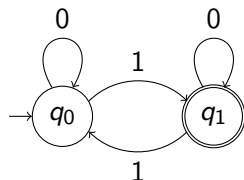
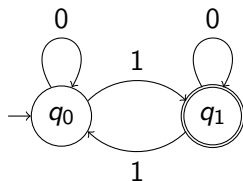


Figure: Transition Diagram of automaton

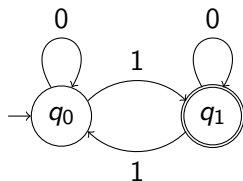
Example: Computation

- On input 1001, the computation is
 - Start in state q_0 . Read 1 and goto q_1 .
 - Read 0 and goto q_1 .
 - Read 0 and goto q_1 .
 - Read 1 and goto q_0 . Since q_0 is not a final state 1001 is **rejected**.



Example: Computation

- On input 1001, the computation is
 - Start in state q_0 . Read 1 and goto q_1 .
 - Read 0 and goto q_1 .
 - Read 0 and goto q_1 .
 - Read 1 and goto q_0 . Since q_0 is not a final state 1001 is **rejected**.
- On input 010, the computation is
 - Start in state q_0 . Read 0 and goto q_0 .
 - Read 1 and goto q_1 .
 - Read 0 and goto q_1 . Since q_1 is a final state 010 is **accepted**.



Example 1

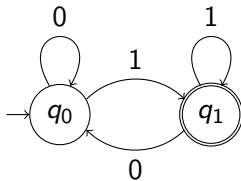


Example 1



Figure: Automaton accepts all strings of 0s and 1s

Example II



Example II

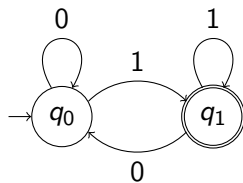
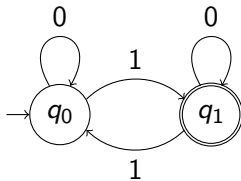


Figure: Automaton accepts strings ending in 1

Example III



Example III

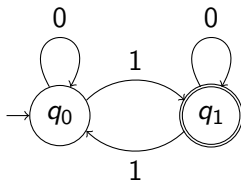
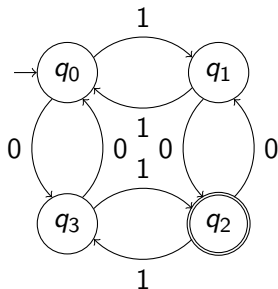


Figure: Automaton accepts strings having an odd number of 1s

Example IV



Example IV

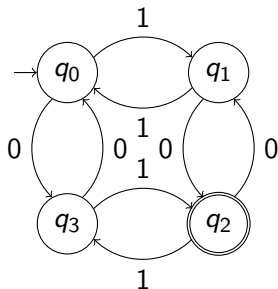


Figure: Automaton accepts strings having an odd number of 1s and odd number of 0s

Finite Automata in Practice

- grep
- Thermostats
- Coke Machines
- Elevators
- Train Track Switches
- Security Properties
- Lexical Analyzers for Parsers

Alphabet

Definition

An **alphabet** is any finite, non-empty set of symbols. We will usually denote it by Σ .

Example

Examples of alphabets include $\{0, 1\}$ (binary alphabet); $\{a, b, \dots, z\}$ (English alphabet); the set of all ASCII characters; $\{\text{moveforward}, \text{moveback}, \text{rotate90}\}$.

Strings

Definition

A **string** or **word** over alphabet Σ is a (finite) sequence of symbols in Σ . Examples are '0101001', 'string'.

Strings

Definition

A **string** or **word** over alphabet Σ is a (finite) sequence of symbols in Σ . Examples are '0101001', 'string'.

- ϵ is the **empty string**.

Strings

Definition

A **string** or **word** over alphabet Σ is a (finite) sequence of symbols in Σ . Examples are '0101001', 'string'.

- ϵ is the **empty string**.
- The **length** of string u (denoted by $|u|$) is the number of symbols in u . Example, $|\epsilon| = 0$, $|011010| = 6$.

Strings

Definition

A **string** or **word** over alphabet Σ is a (finite) sequence of symbols in Σ . Examples are '0101001', 'string'.

- ϵ is the **empty string**.
- The **length** of string u (denoted by $|u|$) is the number of symbols in u . Example, $|\epsilon| = 0$, $|011010| = 6$.
- **Concatenation**: uv is the string that has a copy of u followed by a copy of v . Example, if $u = \text{'cat'}$ and $v = \text{'nap'}$ then $uv = \text{'catnap'}$. If $v = \epsilon$ the $uv = vu = u$.

Strings

Definition

A **string** or **word** over alphabet Σ is a (finite) sequence of symbols in Σ . Examples are '0101001', 'string'.

- ϵ is the **empty string**.
- The **length** of string u (denoted by $|u|$) is the number of symbols in u . Example, $|\epsilon| = 0$, $|011010| = 6$.
- **Concatenation**: uv is the string that has a copy of u followed by a copy of v . Example, if $u = 'cat'$ and $v = 'nap'$ then $uv = 'catnap'$. If $v = \epsilon$ then $uv = vu = u$.
- u is a prefix of v if there is a string w such that $v = uw$. Example 'cat' is a prefix of 'catnap'.

Languages

Definition

Languages

Definition

- For alphabet Σ , Σ^* is the set of all strings over Σ . Σ^n is the set of all strings of length n .

Languages

Definition

- For alphabet Σ , Σ^* is the set of all strings over Σ . Σ^n is the set of all strings of length n .
- A **language** over Σ is a set $L \subseteq \Sigma^*$. For example $L = \{1, 01, 11, 001\}$ is a language over $\{0, 1\}$.

Set Notation

We will often define languages using the set builder notation.
Thus, $L = \{w \in \Sigma^* \mid p(w)\}$ is the collection of all strings w over Σ that satisfy the property p .

Set Notation

We will often define languages using the set builder notation. Thus, $L = \{w \in \Sigma^* \mid p(w)\}$ is the collection of all strings w over Σ that satisfy the property p .

Example

- $L = \{w \in \{0, 1\}^* \mid |w| \text{ is even}\}$ is

Set Notation

We will often define languages using the set builder notation. Thus, $L = \{w \in \Sigma^* \mid p(w)\}$ is the collection of all strings w over Σ that satisfy the property p .

Example

- $L = \{w \in \{0, 1\}^* \mid |w| \text{ is even}\}$ is the set of all even length strings over $\{0, 1\}$.

Set Notation

We will often define languages using the set builder notation. Thus, $L = \{w \in \Sigma^* \mid p(w)\}$ is the collection of all strings w over Σ that satisfy the property p .

Example

- $L = \{w \in \{0, 1\}^* \mid |w| \text{ is even}\}$ is the set of all even length strings over $\{0, 1\}$.
- $L = \{w \in \{0, 1\}^* \mid \text{there is a } u \text{ such that } wu = 10001\}$ is

Set Notation

We will often define languages using the set builder notation. Thus, $L = \{w \in \Sigma^* \mid p(w)\}$ is the collection of all strings w over Σ that satisfy the property p .

Example

- $L = \{w \in \{0, 1\}^* \mid |w| \text{ is even}\}$ is the set of all even length strings over $\{0, 1\}$.
- $L = \{w \in \{0, 1\}^* \mid \text{there is a } u \text{ such that } wu = 10001\}$ is the set of all prefixes of 10001.

Defining an Automaton

To describe an automaton, we need to specify

- What the alphabet is,
- What the states are,
- What the initial state is,
- What states are accepting/final, and
- What the transition from each state and input symbol is.

Thus, the above 5 things are part of the formal definition.

Finite Automata

Formal Definition

Definition

A finite automaton is $M = (Q, \Sigma, \delta, q_0, F)$,
where

- Q is the finite set of states
- Σ is the finite alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ “Next-state” transition function
- $q_0 \in Q$ initial state
- $F \subseteq Q$ final/accepting states

Deterministic Finite Automata

Formal Definition

Definition

A deterministic finite automaton (DFA) is $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is the finite set of states
- Σ is the finite alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ “Next-state” transition function
- $q_0 \in Q$ initial state
- $F \subseteq Q$ final/accepting states

Given a state and a symbol, the next state is “determined”.

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, string $w = w_1 w_2 \cdots w_k$, where for each i $w_i \in \Sigma$, and states $q_1, q_2 \in Q$, we say $q_1 \xrightarrow{w} M q_2$ if there is a sequence of states r_0, r_1, \dots, r_k such that

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, string $w = w_1 w_2 \cdots w_k$, where for each i $w_i \in \Sigma$, and states $q_1, q_2 \in Q$, we say $q_1 \xrightarrow{w} M q_2$ if there is a sequence of states r_0, r_1, \dots, r_k such that

- $r_0 = q_1$,

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, string $w = w_1 w_2 \cdots w_k$, where for each i $w_i \in \Sigma$, and states $q_1, q_2 \in Q$, we say $q_1 \xrightarrow{w} M q_2$ if there is a sequence of states r_0, r_1, \dots, r_k such that

- $r_0 = q_1$,
- for each i , $\delta(r_i, w_{i+1}) = r_{i+1}$, and

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, string $w = w_1 w_2 \cdots w_k$, where for each i $w_i \in \Sigma$, and states $q_1, q_2 \in Q$, we say $q_1 \xrightarrow{w} M q_2$ if there is a sequence of states r_0, r_1, \dots, r_k such that

- $r_0 = q_1$,
- for each i , $\delta(r_i, w_{i+1}) = r_{i+1}$, and
- $r_k = q_2$.

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, string $w = w_1 w_2 \cdots w_k$, where for each i $w_i \in \Sigma$, and states $q_1, q_2 \in Q$, we say $q_1 \xrightarrow{w} q_2$ if there is a sequence of states r_0, r_1, \dots, r_k such that

- $r_0 = q_1$,
- for each i , $\delta(r_i, w_{i+1}) = r_{i+1}$, and
- $r_k = q_2$.

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and string $w \in \Sigma^*$, we say M **accepts** w iff $q_0 \xrightarrow{w} q$ for some $q \in F$.

Acceptance/Recognition

Definition

The **language accepted or recognized** by a DFA M over alphabet Σ is $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Acceptance/Recognition

Definition

The **language accepted or recognized** by a DFA M over alphabet Σ is $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$. A language L is said to be **accepted/recognized** by M if $L = L(M)$.

Acceptance/Recognition and Regular Languages

Definition

The **language accepted or recognized** by a DFA M over alphabet Σ is $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$. A language L is said to be **accepted/recognized** by M if $L = L(M)$.

Definition

A language L is **regular** if there is some DFA M such that $L = L(M)$.

Simple Observations about DFAs

Proposition

For a DFA M , string w , and state q_1 , there is exactly one state q_2 such that $q_1 \xrightarrow{w}_M q_2$.

Simple Observations about DFAs

Proposition

For a DFA M , string w , and state q_1 , there is exactly one state q_2 such that $q_1 \xrightarrow{w}_M q_2$.

Proof.

By induction on $|w|$. □

Simple Observations about DFAs

Proposition

For a DFA M , string w , and state q_1 , there is exactly one state q_2 such that $q_1 \xrightarrow{w}_M q_2$.

Proof.

By induction on $|w|$. □

Proposition

For DFA M , strings u and v , and states q_1 and q_3 , $q_1 \xrightarrow{uv}_M q_3$ if and only if there is a state q_2 such that $q_1 \xrightarrow{u}_M q_2$ and $q_2 \xrightarrow{v}_M q_3$.

Formal Example of DFA

Example

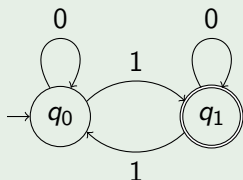


Figure: Transition Diagram of DFA

Formal Example of DFA

Example

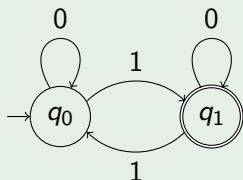


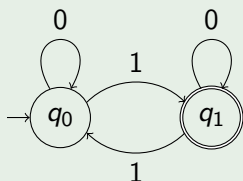
Figure: Transition Diagram of DFA

	0	1
q_0	q_0	q_1
q_1	q_1	q_0

Figure: Transition Table representation

Formal Example of DFA

Example



	0	1
q_0	q_0	q_1
q_1	q_1	q_0

Figure: Transition Table representation

Figure: Transition Diagram of DFA

Formally the automaton is $M_{\text{odd}} = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$
where

$$\delta(q_0, 0) = q_0$$

$$\delta(q_0, 1) = q_1$$

$$\delta(q_1, 0) = q_1$$

$$\delta(q_1, 1) = q_0$$

Language of M_{odd}

Proposition

$L(M_{\text{odd}}) = \{w \in \{0, 1\}^* \mid w \text{ has an odd number of 1s}\}$, where M_{odd} is as defined before.

Proof about the language of M_{odd}

Proof.

We will prove by induction on $|w|$ that $q_0 \xrightarrow{w}_M q_1$ if w has an odd number of 1s.



Proof about the language of M_{odd}

Proof.

We will prove by induction on $|w|$ that $q_0 \xrightarrow{w}_M q_1$ if w has an odd number of 1s.

- **Base Case:** When $w = \epsilon$, w has an even number of 1s and so observation holds vacuously.



Proof about the language of M_{odd}

Proof.

We will prove by induction on $|w|$ that $q_0 \xrightarrow{w}_M q_1$ if w has an odd number of 1s.

- **Base Case:** When $w = \epsilon$, w has an even number of 1s and so observation holds vacuously.
- **Induction Step $w = u0$:** If w has an odd number of 1s then u has an even number of 1s, and so (by induction hypothesis) $q_0 \xrightarrow{u}_M q_0$. Since $q_0 \xrightarrow{0}_M q_1$, $q_0 \xrightarrow{w}_M q_1$.



Proof about the language of M_{odd}

Proof.

We will prove by induction on $|w|$ that $q_0 \xrightarrow{w}_M q_1$ if w has an odd number of 1s.

- **Base Case:** When $w = \epsilon$, w has an even number of 1s and so observation holds vacuously.
- **Induction Step $w = u0$:** If w has an odd number of 1s then u has an even number of 1s, and so (by induction hypothesis) $q_0 \xrightarrow{u}_M q_0$. Since $q_0 \xrightarrow{0}_M q_1$, $q_0 \xrightarrow{w}_M q_1$.
- **Induction Step $w = u1$:** If w has an even number of 1s then to show that M is in q_1 after w , we need to argue that M is in q_0 after u .



Proof about the language of M_{odd}

It fails!

Proof.

We will prove by induction on $|w|$ that $q_0 \xrightarrow{w}_M q_1$ if w has an odd number of 1s.

- **Base Case:** When $w = \epsilon$, w has an even number of 1s and so observation holds vacuously.
- **Induction Step $w = u0$:** If w has an odd number of 1s then u has an even number of 1s, and so (by induction hypothesis) $q_0 \xrightarrow{u}_M q_1$. Since $q_1 \xrightarrow{0}_M q_1$, $q_0 \xrightarrow{w}_M q_1$.
- **Induction Step $w = u1$:** If w has an even number of 1s then to show that M is in q_1 after w , we need to argue that M is in q_0 after u .

Need to prove a stronger statement.



Analyzing the problem

- For the induction step $w = u1$, we need to argue that M is in q_0 after u .
- Proving that if w has an odd number of 1s then M is in state q_1 is not sufficient to show that $L(M_{\text{odd}})$ is the set of strings that has an odd number of 1s

Analyzing the problem

- For the induction step $w = u1$, we need to argue that M is in q_0 after u .
- Proving that if w has an odd number of 1s then M is in state q_1 is not sufficient to show that $L(M_{\text{odd}})$ is the set of strings that has an odd number of 1s because it does not show that **only** strings with an odd number of 1s is accepted!

Corrected Proof

Proof.

We will prove by induction on $|w|$ that after reading w ,
 $q_0 \xrightarrow{w}_M q_1$ if **and only if** w has an odd number of 1s.

Corrected Proof

Proof.

We will prove by induction on $|w|$ that after reading w ,
 $q_0 \xrightarrow{w}_M q_1$ if **and only if** w has an odd number of 1s.

- **Base Case:** When $w = \epsilon$, w has an even number of 1s and M is in state q_0 after w .

Corrected Proof

Proof.

We will prove by induction on $|w|$ that after reading w ,
 $q_0 \xrightarrow{w}_M q_1$ if **and only if** w has an odd number of 1s.

- **Base Case:** When $w = \epsilon$, w has an even number of 1s and M is in state q_0 after w .
- **Induction Step $w = u0$:** w has an odd number of 1s iff u has an odd number of 1s, iff (by ind. hyp.) $q_0 \xrightarrow{u}_M q_1$ iff $q_0 \xrightarrow{w}_M q_1$ (since $q_0 \xrightarrow{0}_M q_0$ and $q_1 \xrightarrow{0}_M q_1$).

Corrected Proof

Proof.

We will prove by induction on $|w|$ that after reading w ,
 $q_0 \xrightarrow{w}_M q_1$ if **and only if** w has an odd number of 1s.

- **Base Case:** When $w = \epsilon$, w has an even number of 1s and M is in state q_0 after w .
- **Induction Step $w = u0$:** w has an odd number of 1s iff u has an odd number of 1s, iff (by ind. hyp.) $q_0 \xrightarrow{u}_M q_1$ iff $q_0 \xrightarrow{w}_M q_1$ (since $q_0 \xrightarrow{0}_M q_0$ and $q_1 \xrightarrow{0}_M q_1$).
- **Induction Step $w = u1$:** w has an odd number of 1s iff u has an even number of 1s iff $q_0 \xrightarrow{u}_M q_0$ (ind. hyp.) iff $q_0 \xrightarrow{w}_M q_1$ (since $q_0 \xrightarrow{1}_M q_1$ and $q_1 \xrightarrow{1}_M q_0$). □

Proving Correctness of a DFA

Proof Template

Given a DFA M having n states $\{q_0, q_1, \dots, q_{n-1}\}$ with initial state q_0 , to prove that $L(M) = L$, we do the following.

Proving Correctness of a DFA

Proof Template

Given a DFA M having n states $\{q_0, q_1, \dots, q_{n-1}\}$ with initial state q_0 , to prove that $L(M) = L$, we do the following.

- 1 Come up with languages L_0, L_1, \dots, L_{n-1} such that $\bigcup_{i=0}^{n-1} L_i = \Sigma^*$ and are pairwise disjoint, i.e., $L_i \cap L_j = \emptyset$ for any $i \neq j$

Proving Correctness of a DFA

Proof Template

Given a DFA M having n states $\{q_0, q_1, \dots, q_{n-1}\}$ with initial state q_0 , to prove that $L(M) = L$, we do the following.

- 1 Come up with languages L_0, L_1, \dots, L_{n-1} such that $\bigcup_{i=0}^{n-1} L_i = \Sigma^*$ and are pairwise disjoint, i.e., $L_i \cap L_j = \emptyset$ for any $i \neq j$
- 2 Prove by induction on $|w|$, $q_0 \xrightarrow{w}_M q_i$ if and only if $w \in L_i$

Proving Correctness of a DFA

Proof Template

Given a DFA M having n states $\{q_0, q_1, \dots, q_{n-1}\}$ with initial state q_0 , to prove that $L(M) = L$, we do the following.

- 1 Come up with languages L_0, L_1, \dots, L_{n-1} such that $\bigcup_{i=0}^{n-1} L_i = \Sigma^*$ and are pairwise disjoint, i.e., $L_i \cap L_j = \emptyset$ for any $i \neq j$
- 2 Prove by induction on $|w|$, $q_0 \xrightarrow{w}_M q_i$ if and only if $w \in L_i$
- 3 Show that $\bigcup_{q_i \in F} L_i = L$; the collection of all strings that reach a final state is **exactly** L .

Typical Problem

Problem

Given a language L , design a DFA M that accepts L , i.e.,
 $L(M) = L$.

Typical Problem

Problem

Given a language L , design a DFA M that accepts L , i.e.,
 $L(M) = L$.

How does one go about it?

Methodology

- Imagine yourself in the place of the machine, reading symbols of the input, and trying to determine if it should be accepted.
- Remember at any point you have only seen a part of the input, and you don't know when it ends.
- Work out what about the string you have seen so far is important. It cannot be all the symbols, and it must be some finite number of bits.

Strings containing 0

Problem

Design an automaton that accepts all strings over $\{0, 1\}$ that contain at least one 0.

Solution

What do you need to remember?

Strings containing 0

Problem

Design an automaton that accepts all strings over $\{0, 1\}$ that contain at least one 0.

Solution

What do you need to remember? Whether you have seen a 0 so far or not!

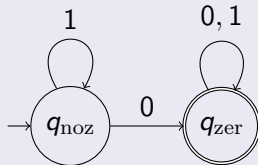


Figure: Automaton accepting strings with at least one 0.

Even length strings

Problem

Design an automaton that accepts all strings over $\{0, 1\}$ that have an even length.

Solution

What do you need to remember?

Even length strings

Problem

Design an automaton that accepts all strings over $\{0, 1\}$ that have an even length.

Solution

What do you need to remember? Whether you have seen an odd or an even number of symbols.

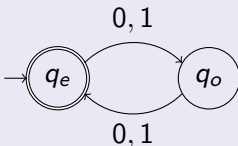


Figure: Automaton accepting strings of even length.

Pattern Recognition

Problem

Design an automaton that accepts all strings over $\{0, 1\}$ that have 001 as a substring, where u is a substring of w if there are w_1 and w_2 such that $w = w_1 u w_2$.

Solution

What do you need to remember?

Pattern Recognition

Problem

Design an automaton that accepts all strings over $\{0, 1\}$ that have 001 as a substring, where u is a substring of w if there are w_1 and w_2 such that $w = w_1 u w_2$.

Solution

What do you need to remember? Whether you

- haven't seen any symbols of the pattern
- have just seen 0
- have just seen 00
- have seen the entire pattern 001

Pattern Recognition Automaton

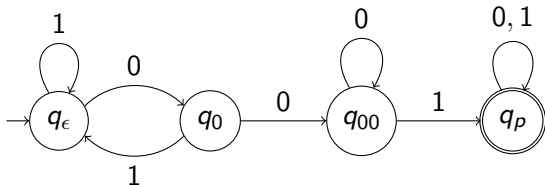


Figure: Automaton accepting strings having 001 as substring.

grep Problem

Problem

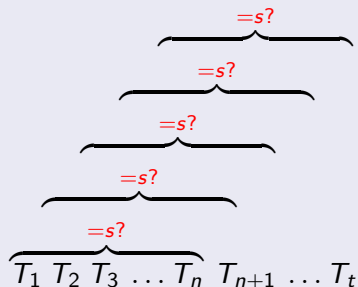
Given text T and string s , does s appear in T ?

grep Problem

Problem

Given text T and string s , does s appear in T ?

Solution

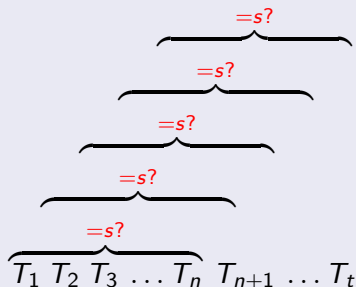


grep Problem

Problem

Given text T and string s , does s appear in T ?

Naïve Solution



Running time = $O(nt)$, where $|T| = t$ and $|s| = n$.

grep Problem

Smarter Solution

Solution

- Build DFA M for $L = \{w \mid \text{there are } u, v. w = usv\}$
- Run M on text T

grep Problem

Smarter Solution

Solution

- Build DFA M for $L = \{w \mid \text{there are } u, v. w = usv\}$
- Run M on text T

Time = time to build M + $O(t)$!

grep Problem

Smarter Solution

Solution

- Build DFA M for $L = \{w \mid \text{there are } u, v. w = usv\}$
- Run M on text T

Time = time to build M + $O(t)$!

Questions

- Is L regular no matter what s is?
- If yes, can M be built “efficiently”?

grep Problem

Smarter Solution

Solution

- Build DFA M for $L = \{w \mid \text{there are } u, v. w = usv\}$
- Run M on text T

Time = time to build M + $O(t)$!

Questions

- Is L regular no matter what s is?
- If yes, can M be built “efficiently”?

Knuth-Morris-Pratt (1977): Yes to both the above questions.

Multiples

Problem

Design an automaton that accepts all strings w over $\{0, 1\}$ such that w is the binary representation of a number that is a multiple of 5.

Solution

What do you need to remember?

Multiples

Problem

Design an automaton that accepts all strings w over $\{0, 1\}$ such that w is the binary representation of a number that is a multiple of 5.

Solution

What do you need to remember? The remainder when divided by 5.

Multiples

Problem

Design an automaton that accepts all strings w over $\{0, 1\}$ such that w is the binary representation of a number that is a multiple of 5.

Solution

What do you need to remember? The remainder when divided by 5.
How do you compute remainders?

Multiples

Problem

Design an automaton that accepts all strings w over $\{0, 1\}$ such that w is the binary representation of a number that is a multiple of 5.

Solution

What do you need to remember? The remainder when divided by 5.

How do you compute remainders?

- If w is the number n then $w0$ is $2n$ and $w1$ is $2n + 1$.
- $(a.b + c) \bmod 5 = (a.(b \bmod 5) + c) \bmod 5$
- e.g. $1011 = 11$ (decimal) $\equiv 1 \bmod 5$
 $10110 = 22$ (decimal) $\equiv 2 \bmod 5$
 $10111 = 23$ (decimal) $\equiv 3 \bmod 5$

Automaton for recognizing Multiples

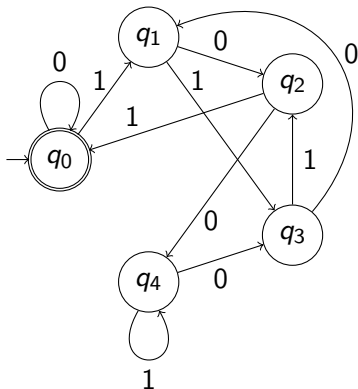


Figure: Automaton recognizing strings encoding binary number that are multiples of 5.

A One k -positions from end

Problem

Design an automaton for the language $L_k = \{w \mid k\text{th character from end of } w \text{ is } 1\}$

Solution

What do you need to remember?

A One k -positions from end

Problem

Design an automaton for the language $L_k = \{w \mid k\text{th character from end of } w \text{ is } 1\}$

Solution

What do you need to remember? The last k characters seen so far!
Formally, $M_k = (Q, \{0, 1\}, \delta, q_0, F)$

- States = $Q = \{\langle w \rangle \mid w \in \{0, 1\}^* \text{ and } |w| \leq k\}$
- $\delta(\langle w \rangle, b) = \begin{cases} \langle wb \rangle & \text{if } |w| < k \\ \langle w_2 w_3 \dots w_k b \rangle & \text{if } w = w_1 w_2 \dots w_k \end{cases}$
- $q_0 = \langle \epsilon \rangle$
- $F = \{\langle 1 w_2 w_3 \dots w_k \rangle \mid w_i \in \{0, 1\}\}$

Lower Bound on DFA size

Proposition

Any DFA recognizing L_k has at least 2^k states.

Lower Bound on DFA size

Proposition

Any DFA recognizing L_k has at least 2^k states.

Proof.

Let M , with initial state q_0 , recognize L_k and assume (for contradiction) that M has $< 2^k$ states.

Lower Bound on DFA size

Proposition

Any DFA recognizing L_k has at least 2^k states.

Proof.

Let M , with initial state q_0 , recognize L_k and assume (for contradiction) that M has $< 2^k$ states.

- Number of string of length $k = 2^k$
- There must be two distinct string w_0 and w_1 of length k such that for some state q , $q_0 \xrightarrow{w_0} q$ and $q_0 \xrightarrow{w_1} q$. □

Proof (contd)

Proof.

Let i be the first position where w_0 and w_1 differ. Without loss of generality assume that w_0 has 0 in the i th position and w_1 has 1.

$$\begin{aligned}w_0 &= \dots 0 \dots \\w_1 &= \underbrace{\dots}_{i-1} 1 \underbrace{\dots}_{k-i}\end{aligned}$$



Proof (contd)

Proof.

Let i be the first position where w_0 and w_1 differ. Without loss of generality assume that w_0 has 0 in the i th position and w_1 has 1.

$$\begin{aligned} w_0 0^{i-1} &= \dots \overbrace{0 \dots 0}^k 0^{i-1} \\ w_1 0^{i-1} &= \underbrace{\dots}_{i-1} 1 \underbrace{\dots}_{k-i} 0^{i-1} \end{aligned}$$



Proof (contd)

Proof.

Let i be the first position where w_0 and w_1 differ. Without loss of generality assume that w_0 has 0 in the i th position and w_1 has 1.

$$\begin{aligned}w_0 0^{i-1} &= \dots \overbrace{0 \dots 0}^k 0^{i-1} \\w_1 0^{i-1} &= \underbrace{\dots}_{i-1} 1 \underbrace{\dots}_{k-i} 0^{i-1}\end{aligned}$$

$w_0 0^{i-1} \notin L_k$ and $w_1 0^{i-1} \in L_k$. Thus, M cannot accept both $w_0 0^{i-1}$ and $w_1 0^{i-1}$. □

Proof (contd)

... Almost there

Proof.

So far, $w_0 0^{i-1} \notin L_n$, $w_1 0^{i-1} \in L_n$, $q_0 \xrightarrow{w_0} q$, and $q_0 \xrightarrow{w_1} q$.



Proof (contd)

... Almost there

Proof.

So far, $w_0 0^{i-1} \notin L_n$, $w_1 0^{i-1} \in L_n$, $q_0 \xrightarrow{w_0} M q$, and $q_0 \xrightarrow{w_1} M q$.

$$q_0 \xrightarrow{w_0 0^{i-1}} M q_1$$



Proof (contd)

... Almost there

Proof.

So far, $w_0 0^{i-1} \notin L_n$, $w_1 0^{i-1} \in L_n$, $q_0 \xrightarrow{w_0} M q$, and $q_0 \xrightarrow{w_1} M q$.

$$q_0 \xrightarrow{w_0 0^{i-1}} M q_1 \quad \text{iff} \quad q \xrightarrow{0^{i-1}} M q_1$$



Proof (contd)

... Almost there

Proof.

So far, $w_0 0^{i-1} \notin L_n$, $w_1 0^{i-1} \in L_n$, $q_0 \xrightarrow{w_0} M q$, and $q_0 \xrightarrow{w_1} M q$.

$$\begin{aligned} q_0 \xrightarrow{w_0 0^{i-1}} M q_1 & \text{ iff } q \xrightarrow{0^{i-1}} M q_1 \\ & \text{ iff } q_0 \xrightarrow{w_1 0^{i-1}} M q_1 \end{aligned}$$



Proof (contd)

... Almost there

Proof.

So far, $w_00^{i-1} \notin L_n$, $w_10^{i-1} \in L_n$, $q_0 \xrightarrow{w_0} M q$, and $q_0 \xrightarrow{w_1} M q$.

$$\begin{aligned} q_0 \xrightarrow{w_00^{i-1}} M q_1 & \text{ iff } q \xrightarrow{0^{i-1}} M q_1 \\ & \text{ iff } q_0 \xrightarrow{w_10^{i-1}} M q_1 \end{aligned}$$

Thus, M accepts or rejects both w_00^{i-1} and w_10^{i-1} .
Contradiction! □