CSE 135: Introduction to Theory of Computation
Deterministic Finite Automata

Sungjin Im

University of California, Merced

01-22-2015

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.

- Example of reduction to a decision problem
  - Given a composite number x, find a factor greater than 1 and less than x.
  - Does x have a factor greater than y and less than z?
- 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): encoded as a string over a finite alphabet

What Does a Computation Look Like?

- Some code (a.k.a. control): the same for all instances
- The input (a.k.a. problem instance): encoded as a string over a finite alphabet
- 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): encoded as a string over a finite alphabet
- 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): encoded as a string over a finite alphabet
- 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): encoded as a string over a finite alphabet
- 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): encoded as a string over a finite alphabet
- 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

  take boolean values (or values in a finite enumerated data type)

Finite State Computation

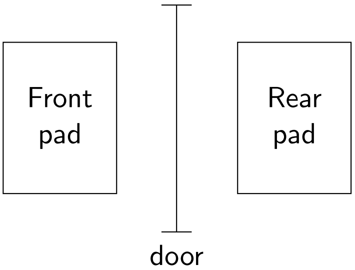
- Finite state: A fixed upper bound on the size of the state, independent of the size of the input
  - take boolean values (or values in a finite enumerated data type)
- Not enough memory to hold the entire input

Finite State Computation

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

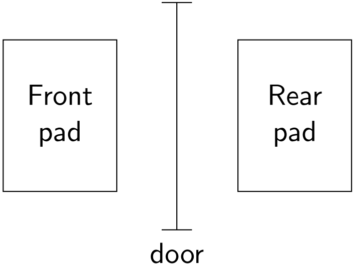
  take boolean values (or values in a finite enumerated data type)
- Not enough memory to hold the entire input
  - "Streaming input": automaton runs (i.e., changes state) on seeing each bit of input

An Automatic Door



Top view of Door

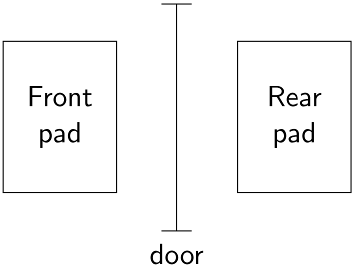
An Automatic Door



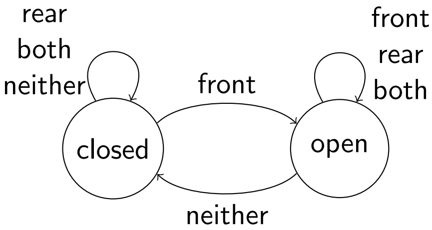
Top view of Door

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

An Automatic Door



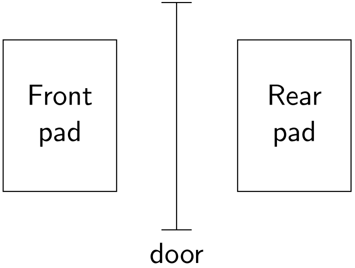
Top view of Door



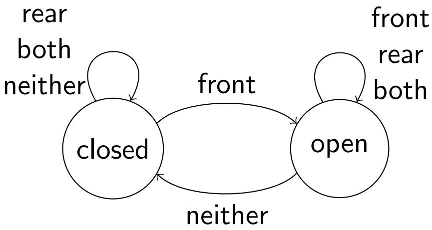
State diagram of controller

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

An Automatic Door



Top view of Door



State diagram of controller

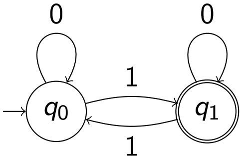
- Input: A stream of events `<front>`, `<rear>`, `<both>`, `<neither>` ...
- Controller has a single bit of state.

Finite Automata

Details

Automaton

A finite automaton has:

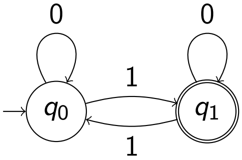


Transition Diagram of automaton

Finite Automata

Automaton

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



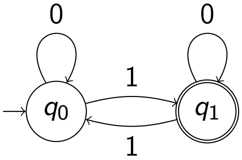
Transition Diagram of automaton

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.



Transition Diagram of automaton

Finite Automata

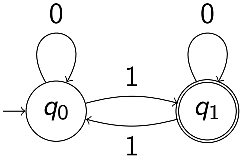
Details

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.



Transition Diagram of automaton

Finite Automata

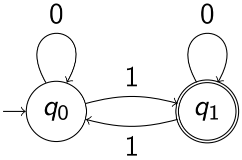
Details

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

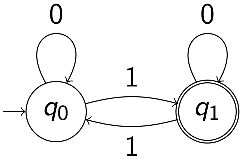
Start at the initial state; in each step, read the next symbol of the input, take the transition (edge) labeled by that symbol to a new state. Acceptance/Rejection: If after reading the input w, the machine is in a final state then w is accepted; otherwise w is rejected.



Transition Diagram of automaton

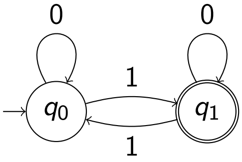
Example: Computation

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

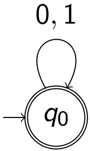


Example: Computation

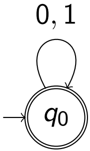
- On input 1001, the computation is
  1. Start in state q_0. Read 1 and goto q_1.
  2. Read 0 and goto q_1.
  3. Read 0 and goto q_1.
  4. Read 1 and goto q_0. Since q_0 is not a final state 1001 is rejected.
- On input 010, the computation is
  1. Start in state q_0. Read 0 and goto q_0.
  2. Read 1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



Example I

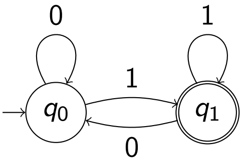


Example I

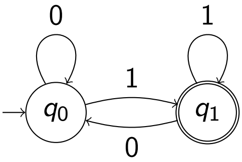


Automaton accepts all strings of 0s and 1s

Example II

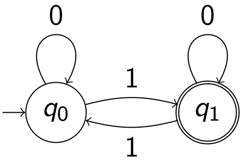


Example II

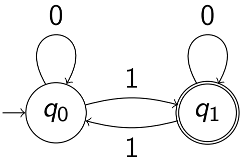


Automaton accepts strings ending in 1

Example III

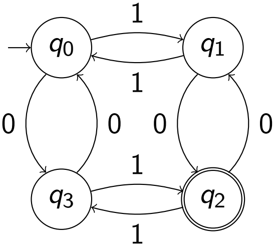


Example III

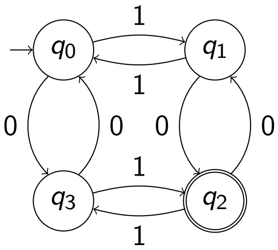


Automaton accepts strings having an odd number of 1s

Example IV



Example IV



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

Finite Automata in Practice

- `grep`: Unix command. grep "word to find" "file name"
- 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); $\{a, b, \ldots, z\}$ (English alphabet); the set of all ASCII characters; $\{\text{moveforward, moveback, rotate90}\}$.

Strings

Definition

A string or word over alphabet Σ is a (finite) sequence of symbols in Σ. Examples are '0101001', 'string', '$\langle \text{moveback} \rangle \langle \text{rotate90} \rangle$'

Strings

Definition

A string or word over alphabet Σ is a (finite) sequence of symbols in Σ. Examples are '0101001', 'string', '$\langle \text{moveback} \rangle \langle \text{rotate90} \rangle$'

- 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', '$\langle\text{moveback}\rangle\langle\text{rotate90}\rangle$'

- 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', '$\langle\text{moveback}\rangle\langle\text{rotate90}\rangle$'

- 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 $v = \epsilon$ 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', '$\langle \mathrm{moveback} \rangle \langle \mathrm{rotate90} \rangle$'

- 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 $uv = vu = u$.
- u is a prefix of v if there is a string w such that $v = uw$. Example 'cat' is a prefix of '$catnap'$.

Strings

Definition

A string or word over alphabet Σ is a (finite) sequence of symbols in Σ. Examples are '0101001', 'string', '$\langle \text{moveback} \rangle \langle \text{rotate90} \rangle$'

- 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 $v = \epsilon$ the $uv = vu = u$.
- u is a prefix of v if there is a string w such that $v = uw$. Example $`cat'$ is a prefix of $`catnap'$.
- Question: Is ϵ 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 $\{0, 1\}$.

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 $\{0, 1\}$.
  - A language L defines a decision problem:

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 $\{0, 1\}$.
  - A language L defines a decision problem: Inputs (strings) whose answer is 'yes' are exactly those belonging to L

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 to 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

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
- $\delta : Q \times \Sigma \to Q$ "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
- $\delta : Q \times \Sigma \to Q$ "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 $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function $\hat{\delta} : Q \times \Sigma^* \to Q$ such that $\hat{\delta}(q, w)$ is M's state after reading w from state q.

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function $\hat{\delta} : Q \times \Sigma^* \to Q$ such that $\hat{\delta}(q, w)$ is M's state after reading w from state q. Formally,

$$\hat{\delta}(q, w) = \begin{cases} & \text{if } w = \epsilon \\ & \text{if } w = ua \end{cases}$$

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function $\hat{\delta} : Q \times \Sigma^* \to Q$ such that $\hat{\delta}(q, w)$ is M's state after reading w from state q. Formally,

$$\hat{\delta}(q, w) = \begin{cases} q & \text{if } w = \epsilon \\ & \text{if } w = ua \end{cases}$$

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function $\hat{\delta} : Q \times \Sigma^* \to Q$ such that $\hat{\delta}(q, w)$ is M's state after reading w from state q. Formally,

$$\hat{\delta}(q, w) = \begin{cases} q & \text{if } w = \epsilon \\ \delta(\hat{\delta}(q, u), a) & \text{if } w = ua \end{cases}$$

Computation

Definition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function $\hat{\delta} : Q \times \Sigma^* \to Q$ such that $\hat{\delta}(q, w)$ is M's state after reading w from state q. Formally,

$$\hat{\delta}(q, w) = \begin{cases} q & \text{if } w = \epsilon \\ \delta(\hat{\delta}(q, u), a) & \text{if } w = ua \end{cases}$$

Definition

We say a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts string $w \in \Sigma^*$ iff $\hat{\delta}(q_0, w) \in F$.

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

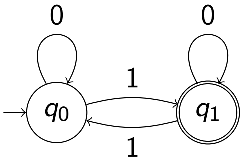
The language accepted or recognized by a DFA M over alphabet Σ is $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$. A language L is said to be accepted/recognized by M if $L = L(M)$.

Definition

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

Formal Example of DFA

Example



Transition Diagram of DFA

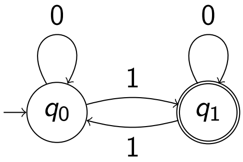
Formally the automaton is $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ where

$$\begin{array}{ll} \delta(q_0, 0) = q_0 & \delta(q_0, 1) = q_1 \\ \delta(q_1, 0) = q_1 & \delta(q_1, 1) = q_0 \end{array}$$

$$\begin{aligned} & \hat{\delta}(q_0, 010) = \delta(\hat{\delta}(q_0, 01), 0) = \delta(\delta(\hat{\delta}(q_0, 0), 1), 0) \\ = \quad & \delta(\delta(\delta(q_0, 0), 1), 0) = \hat{\delta}(\delta(q_0, 0), 10) = \hat{\delta}(q_0, 10) \\ = \quad & \hat{\delta}(\delta(q_0, 1), 0) = \hat{\delta}(q_1, 0) = \delta(q_1, 0) = q_1 \end{aligned}$$

Formal Example of DFA

Example



Transition Diagram of DFA

	0	1
q_0	q_0	q_1
q_1	q_1	q_0

Transition Table representation

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

$$\begin{array}{ll} \delta(q_0, 0) = q_0 & \delta(q_0, 1) = q_1 \\ \delta(q_1, 0) = q_1 & \delta(q_1, 1) = q_0 \end{array}$$

$$\begin{aligned} & \hat{\delta}(q_0, 010) = \delta(\hat{\delta}(q_0, 01), 0) = \delta(\delta(\hat{\delta}(q_0, 0), 1), 0) \\ = \ & \delta(\delta(\delta(q_0, 0), 1), 0) = \hat{\delta}(\delta(q_0, 0), 10) = \hat{\delta}(q_0, 10) \\ = \ & \hat{\delta}(\delta(q_0, 1), 0) = \hat{\delta}(q_1, 0) = \delta(q_1, 0) = q_1 \end{aligned}$$

A Simple Observation about DFAs

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$.

But you can see that this is true by observing that
$\hat{\delta}(q, u_1 u_2 u_3 \ldots u_k) = \delta(\delta(\ldots(\delta(\delta(q, u_1), u_2), u_3), \ldots), u_{k-1}), u_k)$

A Simple Observation about DFAs

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$*, and any strings* $u, v \in \Sigma^*$ *and state* $q \in Q$*,* $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$*.*

Proof.

By induction! Let's see ... $\square$

But you can see that this is true by observing that
$\hat{\delta}(q, u_1 u_2 u_3 \ldots u_k) = \delta(\delta(\ldots(\delta(\delta(q, u_1), u_2), u_3), \ldots), u_{k-1}), u_k)$

Domino Principle

- Line up n dominoes numbered $0, 1, \ldots n-1$ such that if we knock one the next one will fall
- If F_i denotes "ith domino falls", we have $F_i \rightarrow F_{i+1}$

Dominoes

Domino Principle

- Line up n dominoes numbered $0, 1, \ldots n-1$ such that if we knock one the next one will fall
- If F_i denotes "ith domino falls", we have $F_i \rightarrow F_{i+1}$
- Thus, knocking the 0th domino will cause all the dominoes to fall because F_0

Dominoes

Domino Principle

- Line up n dominoes numbered $0, 1, \ldots n-1$ such that if we knock one the next one will fall
- If F_i denotes "ith domino falls", we have $F_i \rightarrow F_{i+1}$
- Thus, knocking the 0th domino will cause all the dominoes to fall because $F_0 \rightarrow F_1$

Dominoes

Domino Principle

- Line up n dominoes numbered $0, 1, \ldots n-1$ such that if we knock one the next one will fall
- If F_i denotes "ith domino falls", we have $F_i \rightarrow F_{i+1}$
- Thus, knocking the 0th domino will cause all the dominoes to fall because $F_0 \rightarrow F_1 \rightarrow F_2$

Dominoes

Domino Principle

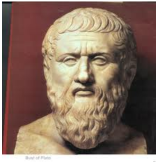
- Line up n dominoes numbered $0, 1, \ldots n-1$ such that if we knock one the next one will fall
- If F_i denotes "ith domino falls", we have $F_i \rightarrow F_{i+1}$
- Thus, knocking the 0th domino will cause all the dominoes to fall because $F_0 \rightarrow F_1 \rightarrow F_2 \rightarrow \cdots \rightarrow F_{n-1}$

Dominoes

Plato's Infinite Domino Principle

Principle

Imagine one domino for each natural number $0, 1, 2, \ldots$, arranged in an infinite row. Knocking the 0th domino will knock them all.

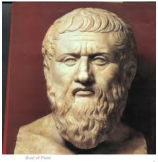


Plato

Plato's Infinite Domino Principle

Principle

Imagine one domino for each natural number $0, 1, 2, \ldots$, arranged in an infinite row. Knocking the 0th domino will knock them all.



Plato

"Proof"

Suppose they don't all fall. Let $k > 0$ be the smallest numbered domino that remains standing. This means domino $k - 1$ fell. But then $k - 1$ will knock k over. Therefore, k must fall and remain standing, which is a contradiction.

Plato's Infinite Domino Principle

Formally

Mathematically we can say

- F_i: ith domino falls
- Suppose for every natural number i, $F_i \to F_{i+1}$
- Suppose 0th domino is knocked over, i.e., F_0
- Then all dominoes will fall, i.e., $\forall i.F_i$.

Dominoes and Mathematical Induction

Domino Principle

- Infinite sequence of dominoes

Induction Principle

- Infinite sequence of statements $S_0, S_1, \ldots$

Dominoes and Mathematical Induction

Domino Principle

- Infinite sequence of dominoes
- Knock the 0th domino

Induction Principle

- Infinite sequence of statements $S_0, S_1, \ldots$
- Prove S_0 is correct [Base Case]

Dominoes and Mathematical Induction

Domino Principle

- Infinite sequence of dominoes
- Knock the 0th domino
- Arrange dominoes such that knocking one will knock the next one

Induction Principle

- Infinite sequence of statements $S_0, S_1, \ldots$
- Prove S_0 is correct [Base Case]
- For an arbitrary i, assuming S_1 to be correct
  establishes S_{i+1} to be correct

Dominoes and Mathematical Induction

Domino Principle

- Infinite sequence of dominoes
- Knock the 0th domino
- Arrange dominoes such that knocking one will knock the next one

Induction Principle

- Infinite sequence of statements $S_0, S_1, \ldots$
- Prove S_0 is correct [Base Case]
- For an arbitrary i, assuming S_1 to be correct [Induction Hypothesis] establishes S_{i+1} to be correct [Induction Step]

Dominoes and Mathematical Induction

Domino Principle

- Infinite sequence of dominoes
- Knock the 0th domino
- Arrange dominoes such that knocking one will knock the next one
- Conclude all dominoes fall

Induction Principle

- Infinite sequence of statements $S_0, S_1, \ldots$
- Prove S_0 is correct [Base Case]
- For an arbitrary i, assuming S_1 to be correct [Induction Hypothesis] establishes S_{i+1} to be correct [Induction Step]
- Conclude $\forall i.\ S_i$ is true

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\hat{\delta}}(q, uv) = \hat{\hat{\delta}}(\hat{\hat{\delta}}(q, u), v)$.

Induction Proofs

An Example

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\hat{\delta}}(q, uv) = \hat{\hat{\delta}}(\hat{\hat{\delta}}(q, u), v)$.

Proof.

We will prove this by induction.

Induction Proofs

An Example

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\hat{\delta}}(q, uv) = \hat{\hat{\delta}}(\hat{\hat{\delta}}(q, u), v)$.

Proof.

We will prove this by induction.

- Let S_i be "$\hat{\hat{\delta}}(q, uv) = \hat{\hat{\delta}}(\hat{\hat{\delta}}(q, u), v)$ when $|v| = i$"

Induction Proofs

An Example

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$*, and any strings* $u, v \in \Sigma^*$ *and state* $q \in Q$*,* $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$*.*

Proof.

We will prove this by induction.

- Let S_i be "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$"
  - Observe that if S_i is true for all i then $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ for every u and v $\cdots\longrightarrow$

Example Inductive Proof

Base Case

Proof (contd).

To establish S_0, i.e., "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = 0$"

Example Inductive Proof

Base Case

Proof (contd).

To establish S_0, i.e., "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = 0$"

- If $|v| = 0$ then $v = \epsilon$
- Observe $u\epsilon = u$
- Thus, LHS $= \hat{\delta}(q, u\epsilon) = \hat{\delta}(q, u)$
- Observe by definition of $\hat{\delta}(\cdot, \cdot)$, for any q', $\hat{\delta}(q', \epsilon) = q'$
- Thus, RHS $= \hat{\delta}(\hat{\delta}(q, u), \epsilon) = \hat{\delta}(q, u)$ $\cdots\!\longrightarrow$

Example Inductive Proof

Induction Step

Proof (contd).

Assume S_i, i.e., "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$". Need to establish S_{i+1}.

$\square$

Example Inductive Proof

Induction Step

Proof (contd).

Assume S_i, i.e., "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$". Need to establish S_{i+1}.

- Consider v such that $|v| = i + 1$.

$\square$

Example Inductive Proof

Induction Step

Proof (contd).

Assume S_i, i.e., "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$". Need to establish S_{i+1}.

- Consider v such that $|v| = i + 1$. WLOG, $v = wa$, where $w \in \Sigma^*$ with $|w| = n$ and $a \in \Sigma$

$\square$

Example Inductive Proof

Induction Step

Proof (contd).

Assume S_i, i.e., "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$". Need to establish S_{i+1}.

- Consider v such that $|v| = i + 1$. WLOG, $v = wa$, where $w \in \Sigma^*$ with $|w| = n$ and $a \in \Sigma$

$$\hat{\delta}(q, uwa) = \delta(\hat{\delta}(q, uw), a) \qquad \text{defn. of } \hat{\delta}$$

□

Example Inductive Proof

Induction Step

Proof (contd).

Assume S_i, i.e., "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$". Need to establish S_{i+1}.

- Consider v such that $|v| = i + 1$. WLOG, $v = wa$, where $w \in \Sigma^*$ with $|w| = n$ and $a \in \Sigma$

$$\begin{aligned} \hat{\delta}(q, uwa) &= \delta(\hat{\delta}(q, uw), a) && \text{defn. of } \hat{\delta} \\ &= \delta(\hat{\delta}(\hat{\delta}(q, u), w), a) && \text{ind. hyp.} \end{aligned}$$

$\square$

Example Inductive Proof

Induction Step

Proof (contd).

Assume S_i, i.e., "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$". Need to establish S_{i+1}.

- Consider v such that $|v| = i + 1$. WLOG, $v = wa$, where $w \in \Sigma^*$ with $|w| = n$ and $a \in \Sigma$

$$
\begin{aligned}
\hat{\delta}(q, uwa) &= \delta(\hat{\delta}(q, uw), a) && \text{defn. of } \hat{\delta} \\
&= \delta(\hat{\delta}(\hat{\delta}(q, u), w), a) && \text{ind. hyp.} \\
&= \hat{\delta}(\hat{\delta}(q, u), wa) && \text{defn. of } \hat{\delta}
\end{aligned}
$$

$\square$

Conventions in Inductive Proofs

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$*, and any strings* $u, v \in \Sigma^*$ *and state* $q \in Q$*,* $\hat{\hat{\delta}}(q, uv) = \hat{\hat{\delta}}(\hat{\hat{\delta}}(q, u), v)$*.*

Proof.

"We will prove by induction on $|v|$" is a short-hand for

Conventions in Inductive Proofs

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$*, and any strings* $u, v \in \Sigma^*$ *and state* $q \in Q$*,* $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$*.*

Proof.

"We will prove by induction on $|v|$" is a short-hand for "We will prove the proposition by induction. Take S_i to be statement of the proposition restricted to strings v where $|v| = i$." □

Properties of $\hat{\delta}$

Corollary

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any string $v \in \Sigma^*$, $a \in \Sigma$ and state $q \in Q$, $\hat{\delta}(q, av) = \hat{\delta}(\delta(q, a), v)$.

Properties of $\hat{\delta}$

Corollary

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any string $v \in \Sigma^*$, $a \in \Sigma$ and state $q \in Q$, $\hat{\delta}(q, av) = \hat{\delta}(\delta(q, a), v)$.

Proof.

From previous proposition we have, $\hat{\delta}(q, av) = \hat{\delta}(\hat{\delta}(q, a), v)$ (taking $u = a$).

$\square$

Properties of $\hat{\delta}$

Corollary

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any string $v \in \Sigma^*$, $a \in \Sigma$ and state $q \in Q$, $\hat{\delta}(q, av) = \hat{\delta}(\delta(q, a), v)$.

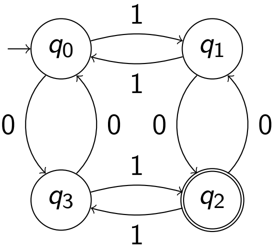
Proof.

From previous proposition we have, $\hat{\delta}(q, av) = \hat{\delta}(\hat{\delta}(q, a), v)$ (taking $u = a$). Next,

$$\begin{aligned} \hat{\delta}(q, a) &= \delta(\hat{\delta}(q, \epsilon), a) && \text{defn. of } \hat{\delta} \\ &= \delta(q, a) && \text{as } \hat{\delta}(q, \epsilon) = q \end{aligned}$$

□

Language of M_{odd}

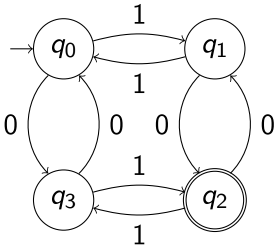


Transition Diagram of M_{odd}

Language of M_{odd}

Proposition

$L(M_{\mathrm{odd}}) = \{w \in \{0,1\}^* \mid w$ has an odd number of 0s and an odd number of 1s$\}$.



Transition Diagram of M_{odd}

Proof about the language of M_{odd}

Proof.

We will prove by induction on $|w|$ that $\hat{\delta}(q_0, w) \in F = \{q_2\}$ iff w has an odd number of 0s and an odd number of 1s.

Proof about the language of M_{odd}

Proof.

We will prove by induction on $|w|$ that $\hat{\delta}(q_0, w) \in F = \{q_2\}$ iff w has an odd number of 0s and an odd number of 1s.

- Base Case: When $w = \epsilon$, w has an even number of 0s and an even number of 1s and $\hat{\delta}(q_0, \epsilon) = q_0$ so the observation holds.

Proof about the language of M_{odd}

Proof.
We will prove by induction on $|w|$ that $\hat{\delta}(q_0, w) \in F = \{q_2\}$ iff w has an odd number of 0s and an odd number of 1s.

- Base Case: When $w = \epsilon$, w has an even number of 0s and an even number of 1s and $\hat{\delta}(q_0, \epsilon) = q_0$ so the observation holds.
- Induction Step $w = 0u$: The parity of the number of 1s in u and w is the same, and the parity of the number of 0s is opposite. And $\hat{\delta}(q_0, w) = \hat{\delta}(\delta(q_0, 0), u) = \hat{\delta}(q_3, u)$

Proof about the language of M_{odd}

It fails!

Proof.

We will prove by induction on $|w|$ that $\hat{\delta}(q_0, w) \in F = \{q_2\}$ iff w has an odd number of 0s and an odd number of 1s.

- Base Case: When $w = \epsilon$, w has an even number of 0s and an even number of 1s and $\hat{\delta}(q_0, \epsilon) = q_0$ so the observation holds.
- Induction Step $w = 0u$: The parity of the number of 1s in u and w is the same, and the parity of the number of 0s is opposite. And $\hat{\delta}(q_0, w) = \hat{\delta}(\delta(q_0, 0), u) = \hat{\delta}(q_3, u)$
- Need to know what strings are accepted from q_3! Need to prove a stronger statement. □

Corrected Proof

Proof.

We need to a stronger statement that asserts what strings are accepted from each state of the DFA. We will prove by induction on $|w|$ that

(a) $\hat{\delta}(q_0, w) \in F$ iff w has odd number of 0s & odd number of 1s

(b) $\hat{\delta}(q_1, w) \in F$ iff

(c) $\hat{\delta}(q_2, w) \in F$ iff

(d) $\hat{\delta}(q_3, w) \in F$ iff

$\cdot\cdot\longrightarrow$

Corrected Proof

Proof.

We need to a stronger statement that asserts what strings are accepted from each state of the DFA. We will prove by induction on $|w|$ that

(a) $\hat{\delta}(q_0, w) \in F$ iff w has odd number of 0s & odd number of 1s
(b) $\hat{\delta}(q_1, w) \in F$ iff w has odd number of 0s & even number of 1s
(c) $\hat{\delta}(q_2, w) \in F$ iff

(d) $\hat{\delta}(q_3, w) \in F$ iff

$\cdot\cdot\!\longrightarrow$

Corrected Proof

Proof.

We need to a stronger statement that asserts what strings are accepted from each state of the DFA. We will prove by induction on $|w|$ that

(a) $\hat{\delta}(q_0, w) \in F$ iff w has odd number of 0s & odd number of 1s
(b) $\hat{\delta}(q_1, w) \in F$ iff w has odd number of 0s & even number of 1s
(c) $\hat{\delta}(q_2, w) \in F$ iff w has even number of 0s & even number of 1s
(d) $\hat{\delta}(q_3, w) \in F$ iff

$\cdots\longrightarrow$

Corrected Proof

Proof.

We need to a stronger statement that asserts what strings are accepted from each state of the DFA. We will prove by induction on $|w|$ that

(a) $\hat{\delta}(q_0, w) \in F$ iff w has odd number of 0s & odd number of 1s

(b) $\hat{\delta}(q_1, w) \in F$ iff w has odd number of 0s & even number of 1s

(c) $\hat{\delta}(q_2, w) \in F$ iff w has even number of 0s & even number of 1s

(d) $\hat{\delta}(q_3, w) \in F$ iff w has even number of 0s & odd number of 1s

$\cdots\!\longrightarrow$

Corrected Proof

Base Case

Proof (contd).

Consider w such that $|w| = 0$. Then $w = \epsilon$.

Corrected Proof

Base Case

Proof (contd).

Consider w such that $|w| = 0$. Then $w = \epsilon$.

- w has even number of 0s and even number of 1s

Corrected Proof

Base Case

Proof (contd).

Consider w such that $|w| = 0$. Then $w = \epsilon$.

- w has even number of 0s and even number of 1s
- For any $q \in Q$, $\hat{\delta}(q, w) = q$

Corrected Proof

Base Case

Proof (contd).

Consider w such that $|w| = 0$. Then $w = \epsilon$.

- w has even number of 0s and even number of 1s
- For any $q \in Q$, $\hat{\delta}(q, w) = q$
- Thus, $\hat{\delta}(q, w) \in F$ iff $q = q_3$, and statements (a),(b),(c), and (d) hold in the base case. $\cdots\!\longrightarrow$

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n.
Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n.

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- Case $q = q_0$, $a = 0$: $\hat{\delta}(q_0, w) \in F$ iff

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- Case $q = q_0$, $a = 0$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- Case $q = q_0$, $a = 0$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff u has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- Case $q = q_0$, $a = 0$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff u has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff w has odd number of 0s and odd number of 1s

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- Case $q = q_0$, $a = 0$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff u has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff w has odd number of 0s and odd number of 1s
- Case $q = q_0$, $a = 1$: $\hat{\delta}(q_0, w) \in F$ iff

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- Case $q = q_0$, $a = 0$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff u has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff w has odd number of 0s and odd number of 1s
- Case $q = q_0$, $a = 1$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_1, u) \in F$ iff

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- Case $q = q_0$, $a = 0$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff u has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff w has odd number of 0s and odd number of 1s
- Case $q = q_0$, $a = 1$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_1, u) \in F$ iff u has odd number of 0s and even number of 1s (by ind. hyp. (b)) iff

Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings w of length n. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length n. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- Case $q = q_0$, $a = 0$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff u has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff w has odd number of 0s and odd number of 1s
- Case $q = q_0$, $a = 1$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_1, u) \in F$ iff u has odd number of 0s and even number of 1s (by ind. hyp. (b)) iff w has odd number of 0s and odd number of 1s

$\cdots\!\longrightarrow$

Corrected Proof

Induction Step: other parts

Proof (contd).

- Case $q = q_1$, $a = 0$: $\hat{\delta}(q_1, w) \in F$ iff $\hat{\delta}(q_2, u) \in F$ iff u has even number of 0s and even number of 1s (by ind. hyp. (c)) iff w has odd number of 0s and even number of 1s

Corrected Proof

Induction Step: other parts

Proof (contd).

- Case $q = q_1$, $a = 0$: $\hat{\delta}(q_1, w) \in F$ iff $\hat{\delta}(q_2, u) \in F$ iff u has even number of 0s and even number of 1s (by ind. hyp. (c)) iff w has odd number of 0s and even number of 1s
- ... And so on for the other cases of $q = q_1$ and $a = 1$, $q = q_2$ and $a = 0$, $q = q_2$ and $a = 1$, $q = q_3$ and $a = 0$, and finally $q = q_3$ and $a = 1$. $\square$

Proving Correctness of a DFA

Proof Template

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

Proving Correctness of a DFA

Proof Template

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

1. Come up with languages $L_0, L_1, \ldots L_{n-1}$ such that $L_0 = L$

Proving Correctness of a DFA

Proof Template

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

1. Come up with languages $L_0, L_1, \ldots L_{n-1}$ such that $L_0 = L$
2. Prove by induction on $|w|$, $\hat{\delta}(q_i, w) \in F$ if and only if $w \in L_i$

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.

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.
- Figure out what to keep in memory. It cannot be all the symbols seen so far: it must fit into a 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?

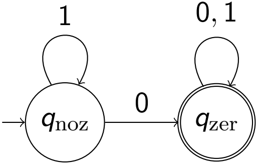
Strings containing 0

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!



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?

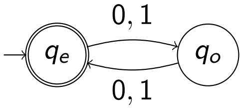
Even length strings

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.



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

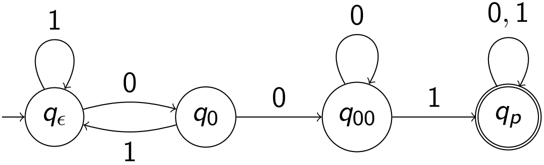
Design an automaton that accepts all strings over $\{0, 1\}$ that have 001 as a substring, where u is a substring of w if there are w_1 and w_2 such that $w = w_1 u w_2$.

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



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

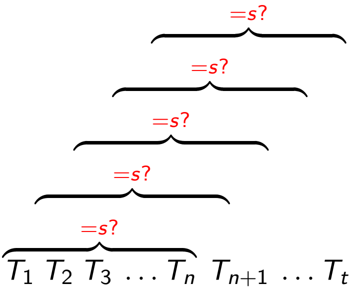
$$\overbrace{T_1\ T_2\ T_3\ \dots\ T_n}^{=s?}\ T_{n+1}\ \dots\ T_t$$

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

grep Problem
Smarter Solution

Solution

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

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

grep Problem

Smarter Solution

Solution

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

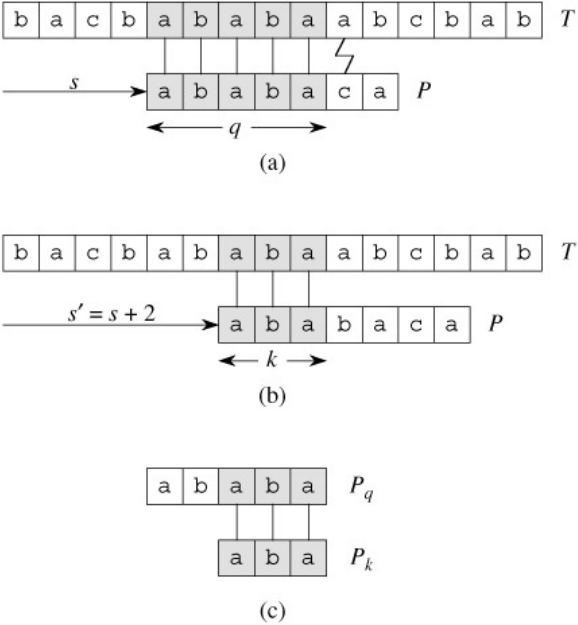
Time $=$ time to build $M + O(t)$!

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.

Knuth-Morris-Pratt (1977)



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 must be remembered?

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 must be remembered? 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 must be remembered? The remainder when divided by 5. How do you compute remainders?

Multiples

Problem

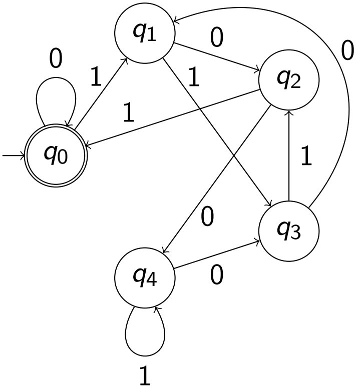
Design an automaton that accepts all strings w over $\{0, 1\}$ such that w is the binary representation of a number that is a multiple of 5.

Solution

What must be remembered? 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



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

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 strings of length $k = 2^k$
- There must be two distinct string w_0 and w_1 of length k such that $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$. $\cdots\!\longrightarrow$

Proof (contd)

Proof (contd).

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 *ith* position and w_1 has 1.

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

$\cdots\!\longrightarrow$

Proof (contd)

Proof (contd).

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 *ith* position and w_1 has 1.

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

$\cdot\cdot\longrightarrow$

Proof (contd)

Proof (contd).

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 *ith* position and w_1 has 1.

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

$w_0 0^{i-1} \notin L_k$ 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 (contd).

So far, $w_0 0^{i-1} \notin L_k$, $w_1 0^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

$\square$

Proof (contd)

. . . Almost there

Proof (contd).

So far, $w_0 0^{i-1} \notin L_k$, $w_1 0^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

$$\hat{\delta}(q_0, w_0 0^{i-1})$$

$\square$

Proof (contd)

. . . Almost there

Proof (contd).

So far, $w_0 0^{i-1} \notin L_k$, $w_1 0^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

$$\hat{\delta}(q_0, w_0 0^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, w_0), 0^{i-1}) \qquad \text{by Proposition proved}$$

$\square$

Proof (contd)

... Almost there

Proof (contd).

So far, $w_0 0^{i-1} \notin L_k$, $w_1 0^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

$$\begin{aligned}
\hat{\delta}(q_0, w_0 0^{i-1}) &= \hat{\delta}(\hat{\delta}(q_0, w_0), 0^{i-1}) && \text{by Proposition proved} \\
&= \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) && \text{by assump. on } w_0 \text{ and } w_1
\end{aligned}$$

$\square$

Proof (contd)

. . . Almost there

Proof (contd).

So far, $w_0 0^{i-1} \notin L_k$, $w_1 0^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

$$\begin{aligned}
\hat{\delta}(q_0, w_0 0^{i-1}) &= \hat{\delta}(\hat{\delta}(q_0, w_0), 0^{i-1}) && \text{by Proposition proved} \\
&= \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) && \text{by assump. on } w_0 \text{ and } w_1 \\
&= \hat{\delta}(q_0, w_1 0^{i-1}) && \text{by Proposition proved}
\end{aligned}$$

$\square$

Proof (contd)

...Almost there

Proof (contd).

So far, $w_0 0^{i-1} \notin L_k$, $w_1 0^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

$$\begin{aligned}
\hat{\delta}(q_0, w_0 0^{i-1}) &= \hat{\delta}(\hat{\delta}(q_0, w_0), 0^{i-1}) && \text{by Proposition proved} \\
&= \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) && \text{by assump. on } w_0 \text{ and } w_1 \\
&= \hat{\delta}(q_0, w_1 0^{i-1}) && \text{by Proposition proved}
\end{aligned}$$

Thus, M accepts or rejects both $w_0 0^{i-1}$ and $w_1 0^{i-1}$.
Contradiction! $\square$