1 Introducing Finite Automata

1.1 Problems and Computation

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 an 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
- 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!

1.2 Finite Automata: Informal Overview

Finite State Computation

- Finite state: A fixed upper bound on the size of the state, independent of the size of the input
  - A sequential program with no dynamic allocation using variables that take boolean values (or values in a finite enumerated data type)
– 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

![An Automatic Door Diagram](image)

Figure 1: Top view of Door

![State Diagram](image)

Figure 2: 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: 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**.
Conventions

- The initial state is shown by drawing an incoming arrow into the state, with no source.
- Final/accept states are indicated by drawing them with a double circle.

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.
1.3 Applications

Finite Automata in Practice

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

2 Formal Definitions

2.1 Deterministic Finite Automaton

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.

Deterministic Finite Automata

Formal Definition

**Definition 1.** A deterministic finite automaton (DFA) is \( M = (Q, \Sigma, \delta, q_0, F) \), where

- \( Q \) is the finite set of states
- \( \Sigma \) is the finite alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \) “Next-state” transition function
Given a state and a symbol, the next state is “determined”.

Formal Example of DFA

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

- $\delta(q_0, 0) = q_0$
- $\delta(q_0, 1) = q_1$
- $\delta(q_1, 0) = q_1$
- $\delta(q_1, 1) = q_0$

Definition 3. 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 $r_0, r_1, \ldots, r_k$ such that

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

Definition 4. For a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and string $w \in \Sigma^*$, we say $M$ accepts $w$ iff $q_0 \xrightarrow{w} M q$ for some $q \in F$. 

Useful Notation
Definition 5. For a DFA \( M = (Q, \Sigma, \delta, q_0, F) \), let us define a function \( \hat{\delta}_M : Q \times \Sigma^* \rightarrow P(Q) \) such that \( \hat{\delta}_M(q, w) = \{ q' \in Q \mid q \xrightarrow{w} M q' \} \).

We could say \( M \) accepts \( w \) iff \( \hat{\delta}_M(q_0, w) \cap F \neq \emptyset \).

Proposition 6. For a DFA \( M = (Q, \Sigma, \delta, q_0, F) \), and any \( q \in Q \), and \( w \in \Sigma^* \), \( |\hat{\delta}_M(q, w)| = 1 \).

Acceptance/Recognition

Definition 7. The language accepted or recognized by a DFA \( M \) over alphabet \( \Sigma \) is \( L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \} \). A language \( L \) is said to be accepted/recognized by \( M \) if \( L = L(M) \).

2.2 Examples

Example I

Example II

Example III
Example IV

![Automaton](image)

Figure 8: Automaton accepts strings having an odd number of 1s

![Automaton](image)

Figure 9: Automaton accepts strings having an odd number of 1s and odd number of 0s

### 3 Designing DFAs

#### 3.1 General Method

**Typical Problem**

Problem

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

**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.
3.2 Examples

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!

\[
\begin{array}{c}
1 \\
\downarrow
\
q_{noz}
\
0, 1
\
\downarrow
\
q_{zero}
\end{array}
\]

Figure 10: 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? Whether you have seen an odd or an even number of symbols.

\[
\begin{array}{c}
0, 1 \\
\downarrow
\
q_{o}
\
0, 1
\
\downarrow
\
q_{e}
\end{array}
\]

Figure 11: 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 \( w = w_1uw_2 \).

Solution
What do you need to remember? Whether you
\begin{itemize}
  \item haven’t seen any symbols of the pattern
\end{itemize}
• have just seen 0
• have just seen 00
• have seen the entire pattern 001

Pattern Recognition Automaton

```
q_e

1
0
q_0

0
q_00

1
q_p
```

Figure 12: Automaton accepting strings having 001 as substring.

**grep Problem**

**Problem**
Given text $T$ and string $s$, does $s$ appear in $T$?

**Naïve Solution**

```
T_1 T_2 T_3 ... T_n T_{n+1} ... T_t
```

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

**grep Problem**

**Smarter Solution**

**Solution**

• Build DFA $M$ for $L = \{ w \mid$ 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.

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 $2n + 1$.
• $(a.b + c) \mod 5 = (a.(b \mod 5) + c) \mod 5$
• e.g. $1011 = 11$ (decimal) $\equiv 1 \mod 5$ $10110 = 22$ (decimal) $\equiv 2 \mod 5$ $10111 = 23$ (decimal) $\equiv 3 \mod 5$

Automaton for recognizing Multiples

![Automaton for recognizing binary numbers that are multiples of 5.](image)

Figure 13: Automaton recognizing binary numbers that are multiples of 5.

A One $k$-positions from end

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

**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 = $Q = \{\langle w \rangle \mid w \in \{0, 1\}^* \text{ and } |w| \leq k\}$
- $\delta(\langle w \rangle, b) = \begin{cases} 
\langle wb \rangle & \text{if } |w| < k \\
\langle w_2w_3\ldots w_kb \rangle & \text{if } w = w_1w_2\ldots w_k 
\end{cases}$
- $q_0 = \langle \epsilon \rangle$
- $F = \{\langle 1w_2w_3\ldots w_k \rangle \mid w_i \in \{0, 1\}\}$