BU CS 332 – Theory of Computation

Lecture 3:
• Deterministic Finite Automata
• Non-deterministic FAs

Reading:
Sipser Ch 1.1-1.2

Mark Bun
February 1, 2021
Last Time

• Parts of a theory of computation: Model for machines, model for problems, theorems relating machines and problems

• Strings: Finite concatenations of symbols
• Languages: Sets $L$ of strings
• Computational (decision) problem: Given a string $x$, is it in the language $L$?
Deterministic Finite Automata
A (Real-Life?) Example

- **Example:** Kitchen scale
- $P =$ Power button (ON / OFF)
- $U =$ Units button (cycles through g / oz / lb)
  
  Only works when scale is ON, but units remembered when scale is OFF

- Starts OFF in g mode

- **A computational problem:** Does a sequence of button presses in $\{P, U\}^*$ leave the scale ON in oz mode?
Machine Models

- **Finite Automata (FAs):** Machine with a finite amount of unstructured memory

Input: \( P \ U \ P \ U \) ... 

Control scans left-to-right
A DFA for the Kitchen Scale Problem
A DFA Recognizing Parity

The language recognized by a DFA is the set of inputs on which it ends in an “accept” state

Parity: Given a string consisting of $a$’s and $b$’s, does it contain an even number of $a$’s?

$Σ = \{a, b\}$  \hspace{1cm} \text{ } \hspace{1cm} \text{ } \hspace{1cm} \text{ } \hspace{1cm} L = \{w \mid w \text{ contains an even number of } a\’s\}$

Which state is reached by the parity DFA on input aabab?

a) “even”

b) “odd”
Anatomy of a DFA
Some Tips for Thinking about DFAs

Given a DFA, what language does it recognize?
- Try experimenting with it on short strings. Do you notice any patterns?
- What kinds of inputs cause the DFA to get trapped in a state?

Given a language, construct a DFA recognizing it
- Imagine you are a machine, reading one symbol at a time, always prepared with an answer
- What is the essential information that you need to remember? Determines set of states.
What language does this DFA recognize?
Practice!

- Lots of worked out examples in Sipser
- Tomorrow’s discussion section
- Automata Tutor: [https://automata-tutor.model.in.tum.de/](https://automata-tutor.model.in.tum.de/)
Formal Definition of a DFA

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

- \( Q \) is the set of states
- \( \Sigma \) is the alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function
- \( q_0 \in Q \) is the start state
- \( F \subseteq Q \) is the set of accept states
A DFA for Parity

Parity: Given a string consisting of $a$’s and $b$’s, does it contain an even number of $a$’s?

$\Sigma = \{a, b\} \quad L = \{w \mid w \text{ contains an even number of } a\’s\}$

State set $Q = \{q_0, q_1\}$
Alphabet $\Sigma = \{a, b\}$
Transition function $\delta$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Start state $q_0$
Set of accept states $F =$
Formal Definition of DFA Computation

A DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w = w_1w_2 \cdots w_n \in \Sigma^*$ (where each $w_i \in \Sigma$) if there exist $r_0, \ldots, r_n \in Q$ such that

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for each $i = 0, \ldots, n - 1$, and
3. $r_n \in F$

$L(M)$ = the language of machine $M$

= set of all strings machine $M$ accepts

$M$ recognizes the language $L(M)$
Example: Computing with the Parity DFA

Let $w = abba$

Does $M$ accept $w$?

A DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w = w_1w_2 \cdots w_n \in \Sigma^*$ (where each $w_i \in \Sigma$) if there exist $r_0, \ldots, r_n \in Q$ such that

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for each $i = 0, \ldots, n - 1$, and
3. $r_n \in F$

What is $\delta(r_2, w_3)$?

- a) $q_0$
- b) $q_1$
Regular Languages

**Definition**: A language is *regular* if it is recognized by a DFA

\[ L = \{ w \in \{a, b\}^* \mid w \text{ has an even number of } a\text{'s} \} \] is regular

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 001} \} \] is regular

Many interesting programs recognize regular languages

- NETWORK PROTOCOLS
- COMPILERS
- GENETIC TESTING
- ARITHMETIC
Let \( TCPS = \{ w \mid \text{w is a complete TCP Session} \} \)

**Theorem.** TCPS is regular
Compilers

Comments:
Are delimited by /* */
Cannot have nested /* */
Must be closed by */
*/ is illegal outside a comment

**COMMENTS** = {strings over {0,1, /, *}} with legal comments

Theorem. **COMMENTS** is regular.
Genetic Testing

DNA sequences are strings over the alphabet \( \{A, C, G, T\} \).

A gene \( g \) is a special substring over this alphabet.

A genetic test searches a DNA sequence for a gene.

\[
\text{GENETICTEST}_g = \{ \text{strings over } \{A, C, G, T\} \text{ containing } g \text{ as a substring} \}
\]

Theorem. \( \text{GENETICTEST}_g \) is regular for every gene \( g \).
**Arithmetic**

**LET** \( \Sigma = \{ [0][0], [0][0][1], [0][0][1], [0][0][1][0], [0][0][1][0][1], [0][0][1][0][1][1] \} \)

- A string over \( \Sigma \) has three ROWS (ROW\(_1\), ROW\(_2\), ROW\(_3\))
- Each ROW \( b_0 b_1 b_2 \ldots b_N \) represents the integer \( b_0 + 2b_1 + \ldots + 2^N b_N \).
- Let ADD = \( \{ S \in \Sigma^* | \text{ROW}_1 + \text{ROW}_2 = \text{ROW}_3 \} \)

**Theorem.** ADD is regular.
Non-deterministic Finite Automata
Non-determinism

In a DFA, the machine is always in exactly one state upon reading each input symbol.

In a nondeterministic FA, the machine can try out many different ways of reading the same string:
- Next symbol may cause an NFA to “branch” into multiple possible computations.
- Next symbol may cause NFA’s computation to fail to enter any state at all.
A Nondeterministic Finite Automaton (NFA) accepts if there \textit{exists} a way to make it reach an accept state.
**Example:** Does this NFA accept the string 1100?
Some special transitions

0

0, 1

0, 1

ε

1
Example

$L(M) =$

\[ L(M) = 0,1 \]

\[ L(M) = \]
Example

0,1

1

0,ε

1

0,1

Diagram of a deterministic finite automaton (DFA) with transitions labeled by inputs 0, 1, and the null string (ε). The diagram starts with an input 0,1, followed by a transition on input 1 to a state labeled 0,ε, then to a final state labeled 1, and finally a loop on input 0,1.
Now You Try!

What is the language of this NFA? (over alphabet \{0\})

a) \{0^k \mid k \text{ is a multiple of } 2\}
b) \{0^k \mid k \text{ is a multiple of } 3\}
c) \{0^k \mid k \text{ is a multiple of } 6\}
d) \{0^k \mid k \text{ is a multiple of } 2 \text{ or a multiple of } 3\}
Formal Definition of a NFA

An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

- $Q$ is the set of states
- $\Sigma$ is the alphabet
- $\delta: Q \times \Sigma \varepsilon \rightarrow P(Q)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

$M$ accepts a string $w$ if there exists a path from $q_0$ to an accept state that can be followed by reading $w$. 
Example

$M = (Q, \Sigma, \delta, Q_0, F)$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$F = \{q_4\}$

$\delta(q_2, 1) =$

$\delta(q_3, 1) =$
Example

\[ N = (Q, \Sigma, \delta, q_0, F) \]
\[ Q = \{q_0, q_1, q_2, q_3\} \]
\[ \Sigma = \{0, 1\} \]
\[ F = \{q_3\} \]

\[ \delta(q_0, 0) = \]
\[ \delta(q_0, 1) = \]
\[ \delta(q_1, \varepsilon) = \]
\[ \delta(q_2, 0) = \]
Nondeterminism

Ways to think about nondeterminism

- (restricted) parallel computation
- tree of possible computations
- guessing and verifying the “right” choice
Why study NFAs?

• Not really a realistic model of computation: Real computing devices can’t actually try many possibilities in parallel

But:

• Useful tool for understanding power of DFAs/regular languages
• NFAs can be simpler than DFAs
• Lets us study “nondeterminism” as a resource (cf. P vs. NP)