Foundations of Computer Science
Lecture 24

Deterministic Finite Automata (DFA)

A Simple Computing Machine: A CPU with States and Transitions
What Problems Can It Solve: Regular Languages
Is There A Problem It Can’t Solve?
A problem can be harder in two ways.

1. The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.

2. The problem needs a different kind of computing machine, with superior capabilities.

The first type of “harder” is the focus of a follow-on algorithms course.

We focus on what can and can’t be solved on a particular kind of machine.
A simple computing machine.
- States.
- Transitions.
- No scratch paper.

What computing problems can this simple machine solve?
- Vending machine.

Regular languages.
- Closed under all the set operations: union, intersection, complement, concatenation, Kleene-star.

Are there problems that cannot be solved?
A Simple Computing Machine

1: Process the input string (left-to-right) starting from the initial state \( q_0 \).
2: Process one bit at a time, each time transitioning from the current state to the next state according to the transition instructions.
3: When done processing every bit, output \( \text{YES} \) if the final resting state of the DFA is a \( \text{YES} \)-state; otherwise output \( \text{NO} \).
Running the Machine on an Input

\[ \begin{array}{c}
0 & 1 & 0 \\
\uparrow & & \\
q_0 & 0 & q_1 \\
& 1 & \downarrow 1 \\
& & q_2 \\
& & 0,1 \\
\end{array} \]

\[ q_0 \mapsto 010 \]

\[ \begin{array}{c}
0 & 1 & 0 \\
\uparrow & & \\
q_0 & 0 & q_1 \\
& 0 & \downarrow 1 \\
& & q_2 \\
& & 0,1 \\
\end{array} \]

\[ q_0 \mapsto 010 \quad \Rightarrow \quad q_1 \mapsto 010 \]

\[ (M \text{ is the name of our “Machine”}) \]

\[ \begin{array}{c}
0 & 1 & 0 \\
\uparrow & & \\
q_0 & 0 & q_1 \\
& 1 & \downarrow 1 \\
& & q_2 \\
& & 0,1 \\
\end{array} \]

\[ q_0 \mapsto 010 \quad \Rightarrow \quad q_1 \mapsto 010 \]

\[ q_2 \mapsto 010 \]

\[ \begin{array}{c}
0 & 1 & 0 \\
\uparrow & & \\
q_0 & 0 & q_1 \\
& 1 & \downarrow 1 \\
& & q_2 \\
& & 0,1 \\
\end{array} \]

\[ q_0 \mapsto 010 \quad \Rightarrow \quad q_2 \mapsto 010 \]

\[ q_2 \mapsto 010 \]

\[ \text{[NO, REJECT]} \]

**Pop Quiz.** Give computation trace for ε, 010, 000. What strings does the machine ACCEPT and say [YES]?

**Pop Quiz.** Determine [YES] or [NO] if you can from partial traces. \( q_0 \mid \square \Rightarrow 0000; \ q_1 \mid \square \Rightarrow 0000; \ q_2 \mid \square \Rightarrow 0000. \)
Computing Problem Solved by a DFA

The computing problem solved by $M$ is the language $\mathcal{L}(M) = \{ w \mid M(w) = \text{YES} \}$. 

$\mathcal{L}(M)$ is the automaton’s $\text{YES}$-set. For our automaton $M$

$$\mathcal{L}(M) = \{0, 00, 000, 0000, \ldots\} = \{0^n \mid n > 0\}.$$ 

1. For an automaton $M$, what is the computing problem $\mathcal{L}(M)$ solved by $M$?

2. For a computing problem $\mathcal{L}$, what automaton $M$ solves $\mathcal{L}$, i.e., $\mathcal{L}(M) = \mathcal{L}$?

Practice. Exercise 24.2 gives you lots of training in question 1.
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

Input sequence: 10¢, 10¢, 5¢, 10¢, 10¢, 10¢.

0¢ → 10¢ → 20¢ → 0¢ (+ soda) → 10¢ → 20¢ → 5¢ (+ soda).
DFA for a Finite Language

\[ \mathcal{L} = \{10\}. \]

- 0 means move to a rejecting ERROR state and stay there
- 1 is partial success.
- Another 1 puts you into ERROR since you want 0;
- 0 from \( q_1 \) and you are ready to accept ... unless ...
- More bits arrive, in which case move to ERROR.

**Practice.** Try random strings other than 01 and make sure our DFA rejects them.
$L_1 = \ast 0 \ast$
$= \{\text{strings with a 0}\}$
$= \{0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots\}$

$M_1$

$L_2 = \ast 1$
$= \{\text{strings ending in 1}\}$
$= \{1, 01, 11, 001, 011, 101, 111, \ldots\}$

$M_2$

**Complement.** Consider $\overline{L_1}$: Must accept strings $M_1$ rejects.

$M$

← flip **YES** and **NO**-states.
Two DFAs in One: Union and Intersection

\[ L_1 = \ast 0 \ast \]

\[ L_2 = \ast 1 \]

\( \ast = \Sigma^* \)

\( M_1 \)

\[ q_0 \xrightarrow{0} q_1 \]

\[ q_0 \xrightarrow{1} q_0 \]

\[ q_1 \xrightarrow{0} q_1 \]

\[ q_1 \xrightarrow{1} q_1 \]

\( M_2 \)

\[ s_0 \xrightarrow{0} s_0 \]

\[ s_0 \xrightarrow{1} s_1 \]

\[ s_1 \xrightarrow{1} s_1 \]

The Joint-DFA has product states \( \{ q_0s_0, q_0s_1, q_1s_0, q_1s_1 \} \):

\[ q_0s_0: M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_0. \]

\[ q_0s_1: M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_1. \]

\[ q_1s_0: M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_0. \]

\[ q_1s_1: M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_1. \]

Pop Quiz.

1. Run the joint and individual DFAs for \( \varepsilon, 0100, 11, 101 \). What are the final states of each DFA?
2. If you want to solve \( L_1 \cup L_2 \), what should the accept states of the joint-DFA be?
3. If you want to solve \( L_1 \cap L_2 \), what should the accept states of the joint-DFA be?
Concatenation and Kleene Star

\[ L_1 = \{1\} \quad (M_1) \]

\[ L_2 = \{0\} \quad (M_2) \]

\[ L_1 \cdot L_2: \]

\[ L_1^*: \]
The Power of DFAs: What can they Solve?

- Finite languages.
  (building blocks of regular expressions)

- Complement, intersection, union.
  (operations to form complex regular expressions)

- Concatenation and Kleene-star (little more complicated, see text).
  (operations to form complex regular expressions)

That’s what we need for regular expressions.

DFAs solve languages (computing problems) expressed as regular expressions.

(That is why the languages solved by DFAs are called regular languages.)
Is There Anything DF As Can’t Solve?

**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
\mathcal{L}_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.
\]

**Theorem.** There is no DFA that solves \( \mathcal{L}_{0^n1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)
\]

After \( k \) 0’s, \( k + 1 \) states visited. There must be a repetition (pigeonhole).

\[
\text{state}(0^i) = \text{state}(0^j) = q, \quad i < j \leq k.
\]

Consider the two input strings \( 0^i1^i \in \mathcal{L}_{0^n1^n} \) and \( 0^j1^i \notin \mathcal{L}_{0^n1^n} \).

After \( M \) has processed the 0s in both strings, it is in state \( q \), and the traces of the two computations are

\[
q \xrightarrow{0^i} 1^i \quad \text{and} \quad q \xrightarrow{0^j} 1^i.
\]

Same number of 1’s remain, from state \( q \). Either both rejected or both accepted. **FISHY!**

**Intuition:** The DFA has no “memory” to remember \( n \).
DFAs can be implemented using basic technology, so practical.

Powerful (regular languages), but also limited.

DFAs fail at so simple a problem as equality.
- That’s not acceptable.
- We need a more powerful machine.
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:

- see the top sheet;
- remove the top sheet \((pop)\)
- \(push\) something new onto the top of the stack.
- no access to inner sheets without removing top.

DFA with a stack is a pushdown automaton \((PDA)\)

How does the stack help to solve \(\{0^n1^n \mid n \geq 0\}\)?

1: When you read in each 0, write it to the stack.
2: For each 1, pop the stack. At the end if the stack is empty, ACCEPT.

The memory allows the automaton to “remember” \(n\).