Formal Definition of DFA

Described by a 5-tuple:

\[ M = (Q, \Sigma, \delta, q_0, F) \]

- \( Q = \) Set of States
  - Finite Number of states
- \( \Sigma = \) Alphabet, a Finite Set of Symbols
- \( \delta = \) The TRANSITION FUNCTION
  - \( \delta : Q \times \Sigma \rightarrow Q \)
- \( q_0 = \) The STARTING STATE
  - \( q_0 \in Q \) (or “INITIAL” STATE)
- \( F = \) The set of ACCEPT states
  - (or “FINAL” STATES) \( F \subseteq Q \)

by default, function means “total” function!
Alternative Definition of Transition

If we change the transition function $\delta$ from a total function to a **partial** function then we don’t need to include trap state because whenever $\delta(s, a)$ is undefined, it goes to the trap state.

Note that the definition of computation remains unchanged.

https://en.wikipedia.org/wiki/Partial_function
Formal Definition of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$

Let $w = w_1 w_2 \ldots w_n$ be a string
where $w_i \in \Sigma$

$M$ accepts $w$ iff there is a sequence of states
$r_0, r_1, r_2, \ldots, r_n$ in $Q$
such that
$r_0 = q_0$
$\delta(r_i, w_{i+1}) = r_{i+1}$ for $0 \leq i < n$
$r_n \in F$

We say...

$M$ “recognizes” Language $A$
if $A = \{ w | M \text{ accepts } w \}$
Redefine computation: Extend $\delta$ to $\delta^*$

Defining the computation of an FA $M = (Q, \Sigma, q_0, A, \delta)$.

Extended transition function $\delta^* : Q \times \Sigma^* \rightarrow Q$:

1) For every $q \in Q$, let $\delta^*(q, \Lambda) = q$

2) For every $q \in Q$, $y \in \Sigma^*$, and $\sigma \in \Sigma$, let $\delta^*(q, y\sigma)$

We say that a string $x \in \Sigma^*$ is **accepted by $M$** iff, $\delta^*(q_0, x) \in A$.

$$\delta^*(q, w) = \begin{cases} \delta(\delta^*(q, x), a) & \text{where } w = xa \text{ and } x \in \Sigma^*, a \in \Sigma \\ q & \text{where } w = \epsilon \end{cases}$$

$$L((Q, \Sigma, \delta, q_0, F)) = \{ w \in \Sigma^* | \delta^*(q_0, w) \in F \}$$

Q1: what about decomposing $w = ax$ or even $w = xy$?

Q2: what about partial function $\delta$?

$\delta^*$ is not defined in Sipser, but is in all other textbooks. This is probably one of the biggest flaws of Sipser book.
Language, String, Machine

**The Empty String**
\[ \varepsilon \text{ (epsilon, } \varepsilon) \]

**The Empty Language**
\[ \emptyset = \varepsilon \varepsilon \]

**Note:**
\[ \varepsilon \varepsilon \neq \emptyset \]
\[ \varepsilon \neq \emptyset \]

---

- does M accept any string?
- does M accept empty string?
- does M recognize the empty language?

---

If a machine accepts NO strings then it recognizes the **Empty Language**.
What’s the language of ...
Q1: language over \{a,b,c\} s.t. the # of a's is divisible by 3

hint: need 3 states:
state 0: (#a's) % 3 == 0
state 1: (#a's) % 3 == 1
state 2: (#a's) % 3 == 2

wait... what about b/c?

Q2: language over \{a,b,c\} s.t. (the # of a's) - (the # of b's) is divisible by 3

hint: still 3 states:
state 0: ((#a's) - (#b's)) % 3 == 0
state 1: ((#a's) - (#b's)) % 3 == 1
state 2: ((#a's) - (#b's)) % 3 == 2
Binary Numbers Divisible by 3

- still just need 3 states
- decimal num divisible by 3?
- again, 3 states
- general strategy in base $d$
  - divisible by $n \Rightarrow n$ states
  - $[0] [1] \ldots [n-1]$
    - $[i] = \{ x \mid x \% n = i \}$
    - at state $[q]$, on digit $i$
      - goto state $[(q \times d + i) \% n]$
  - but it’s possible to use less than $n$ states!
Construct Finite Automaton for ...

- any string that does not contain 001 in it
- try simpler problem: a string that does contain 001

This reminds you of KMP string matching

https://www.ics.uci.edu/~eppstein/161/960222.html
Construct Finite Automaton for ...

- any string that does contain 10111 in it

The general strategy for “contains pattern $a_0a_1...a_{n-1}$”:
- **backbone**: at state $i$ on $a_i$ goto state $i+1$ $(n+1$ states$)$
- **deviations**: at state $i$, on any input $b \neq a_i$
  - go back to the rightmost such state $j$,
  - where prefix $a_0a_1...a_{j-1} = \text{suffix } a_{i-j+1}...a_{i-1}b$
  - because we can reuse such suffix and
  - don’t need to restart from the very beginning

https://www.ics.uci.edu/~eppstein/161/960222.html
Complement Language

Notation

$L(M_1) = \text{The language that } M_1 \text{ recognizes.}$
$L(M_1) = \{x \mid x \in \Sigma^*, \text{ that contain } 0011 \text{ as a substring.}\}$

$L(M_2) = \text{The set of strings over } \{0,1\}^*$
$L(M_2) = \text{that do not contain } 0011.$

Complimenting a Language

They are sets, after all.
$L(M_1) = L(M_2)$

The "Universe"
All possible strings made with symbols from the alphabet.
$\Sigma = \{0,1\}^*$ Universe = $\{0,1\}^*$
Set complement is always relative to some Universe; (implicitly).

$L \triangleq \Sigma^* \setminus L = \{x \mid x \in \Sigma^*, x \notin L\}$
if $L = L(M)$ and $M = (Q, \Sigma, \delta, q, F)$ then
$L = L(\overline{M})$ where $\overline{M} = (Q, \Sigma, \delta, q, \overline{F})$
where $\overline{F} = Q \setminus F$

just flip final and non-final!
Prove: Complement of Regular is Regular

**Proof:** For every regular language $L$, by definition of regular language, there must be a DFA $M$ s.t. $L(M) = L$. let $M = (Q, \Sigma, \delta, q_0, F)$ where $\delta$ is a **total** function, let us construct another DFA $\overline{M} = (Q, \Sigma, \delta, q_0, \overline{F})$ where $\overline{F} = Q \setminus F$.

For every string $w \in L$, it will end up in a state $q \in F$ in $M$, and it will end up in the same state in $\overline{M}$ which rejects $w$ since $q \notin \overline{F}$; similarly, for every string $w'$ in the complement language, i.e., $w \in \Sigma^* \setminus \overline{F}$, it will end up in a state $q' \notin \overline{F}$ in $M$, and it will end up in the same state in $\overline{M}$ which accepts $w$ since $q' \in \overline{F}$. So $\overline{M}$ accepts all strings in $\Sigma^* \setminus L$ **and only those**, which means the complement language $\Sigma^* \setminus L$ is recognized by DFA $\overline{M}$, thus regular. \(\square\)

Note, however, that if $\delta$ is a partial function (i.e., trap state omitted), this proof does not work (why?). You would have to add a trap state and all trap transitions to make $\delta$ a total function first.
Rewrite/Simplify using $\delta^*$ notation

**Proof:** For every regular language $L$, by the definition of regular language, there must be a DFA $M$ s.t. $L(M) = L$. Let $M = (Q, \Sigma, \delta, q_0, F)$ where $\delta$ is a total function, we construct another DFA $\overline{M} = (Q, \Sigma, \delta, q_0, \overline{F})$ where $\overline{F} = Q \setminus F$.

For every string $w \in L$, there exists $q \in Q$ s.t. $\delta^*(q_0, w) = q \in F$ in $M$, and $\delta^*(q_0, w) = q \notin \overline{F}$ in $\overline{M}$ which rejects $w$; similarly, for every string $w'$ in the complement language, i.e., $w' \in \Sigma^* \setminus F$, there exists $q' \in Q$ s.t. $\delta^*(q_0, w') = q' \notin F$ in $M$, and $\delta^*(q_0, w') = q' \in \overline{F}$ in $\overline{M}$ which accepts $w$. So $\overline{M}$ accepts all strings in $\Sigma^* \setminus L$ and nothing else, which means the complement language $\Sigma^* \setminus L$ is recognized by DFA $\overline{M}$, thus regular.

\[
\delta^*(q, w) = \begin{cases} 
\delta(\delta^*(q, x), a) & \text{where } w = xa \text{ and } x \in \Sigma^*, a \in \Sigma \\
q & \text{where } w = \epsilon 
\end{cases}
\]

\[
L((Q, \Sigma, \delta, q_0, F)) = \{ w \in \Sigma^* | \delta^*(q_0, w) \in F \}\]
Binary Number Divisible by 4

- 4-state solution (trivial)
- 3-state solution (merge q1 w/ q3)
- in general, how do you:
  - reduce a DFA to a smaller but equivalent DFA?
    - see Linz 2.4 or Sipser problem 7.42 (p. 327)
    - will discuss later after NFA
  - test if two DFAs are equivalent?
    - follow pairs of states, check if all visited state-pairs agree on finality (both accept or both reject)
    - $O(n^2 \Sigma)$ time and space

https://www.cse.iitb.ac.in/~trivedi/courses/cs208-spring14/lec05.pdf
Test if two DFAs are equivalent

- traverse all state-pairs and make sure each pair agrees on “finality” (both accept or both reject)

<table>
<thead>
<tr>
<th>pair</th>
<th>final?</th>
<th>on 0</th>
<th>on 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A, A)</td>
<td>(y, y)</td>
<td>(B, B)</td>
<td>(A, C)</td>
</tr>
<tr>
<td>(B, B)</td>
<td>(n, n)</td>
<td>(A, C)</td>
<td>(B, B)</td>
</tr>
<tr>
<td>(A, C)</td>
<td>(y, y)</td>
<td>(B, B)</td>
<td>(A, A)</td>
</tr>
</tbody>
</table>

no new pairs found
Proof by Induction  (Linz 1.1-1.2, Sipser 0.4)

• Theorem to prove: $|uv| = |u| + |v|$

• first define string length rigorously and inductively:
  • $|a| = 1$, $|\varepsilon| = 0$, $|wa| = |w| + 1$

• now prove the Theorem by induction on $|v|$
  • base case: $|v| = 0$, then $v = \varepsilon$, so $|uv| = |u| = |u| + 0 = |u| + |v|$
  • inductive case: assume Theorem holds for any $|v|$ of length $0...n$
    Now take any $v$ of length $n+1$. Let $v = wa$, then $|v| = |w| + 1$ (by def.)
    • then $|uv| = |uwa| = |uw| + 1$ (by definition)
    • by induction hypothesis (applicable to any $w$ of length $n$)
      • $|uw| = |u| + |w|$, so that $|uv| = |u| + |w| + 1 = |u| + |v|$

HW: prove by induction on $|u|$
What’s wrong with this proof?

**Theorem(?!):** All horses are the same color.

**Proof:** Let $P(n)$ be the predicate “in all non-empty collections of $n$ horses, all the horses are the same color.” We show that $P(n)$ holds for all $n$ by induction on $n$ (using 1 as the base case).

**Base case:** Clearly, $P(1)$ holds.

**Induction case:** Given $P(n)$, we must show $P(n + 1)$.

Consider an arbitrary collection of $n + 1$ horses. Remove one horse temporarily. Now we have $n$ horses and hence, by the induction hypothesis, these $n$ horses are all the same color. Now call the exiled horse back and send a different horse away. Again, we have a collection of $n$ horses, which, by the induction hypothesis, are all the same color. Moreover, these $n$ horses are the same color as the first collection. Thus, the horse we brought back was the same color as the second horse we sent away, and all the $n + 1$ horses are the same color.
Quiz 1 scores and Projected Final Grade

- mean: 2.0, median: 2

Projected final grade:

- 0: F
- 0.5: C/C-
- 1: C+
- 1.5: B-
- 2: B
- 2.5: B+
- 3: A-
- 3.5: A
- 4: A
- 4.5: A
- 5:

CS 321 - TOC
Regular Language

**Definition**

A language is a **regular language** iff some finite state machine recognizes it.

What languages are NOT regular?

Anything that requires memory. The F.S.M. memory is very limited. Cannot store the string. Cannot "count."

Not Regular:

- \( \{ w^2 \mid w \in \Sigma^* \} \)

- \( \{ w^1 \mid w \in \Sigma^* \} \)

Imagine a string from here to the Moon. You are trying to recognize it. Your only memory: A single small number (i.e., # of states).

...abacababac...
Regular Operations

- other operations:
  - intersection
  - complement
  - difference

- regular languages are closed under all these operations

UNION
\[ A \cup B = \{ x \mid x \in A \text{ or } x \in B \} \]

CONCATENATION
\[ A \circ B = \{ xy \mid x \in A \text{ and } y \in B \} \]

STAR "closure"
\[ A^* = \{ x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A \} \]

EXAMPLE
\[ \Sigma = \{ a, b, c, \ldots \} \]
\[ A = \{ aa, b \} \]
\[ B = \{ x, yy \} \]

\[ A \cup B = \{ aa, b, x, yy \} \]
\[ A \circ B = \{ aax, aayy, bx, byy \} \]
\[ A^* = \{ \epsilon, aa, b, aaaa, aab, baa, bb, \ldots \} \]
\[ aaaaaa, aaab, aabb, aabb \ldots \]