L2.

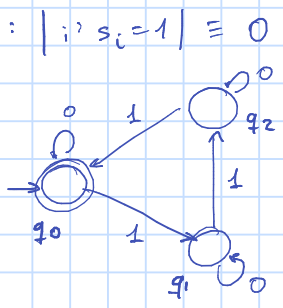
Automata

Informally: state machine, only memory is current state.

Move between states with each character from input.

Represent with state diagram.

e.g. for $\{s : |i : s_i = 1| \equiv 0 \bmod 3\}$



Formally: 4-tuple $M = (Q, \Sigma, \delta, q_0, F)$

- Q: states (finite)
- Σ: alphabet (finite)
- δ: $Q \times \Sigma \to Q$ transition function
- $q_0 \in Q$: initial state
- $F \subseteq Q$: accepting states.

e.g. $Q = \{q_0, q_1, q_2\}$. M_3
$\Sigma = \{0, 1\}$

$\delta =$

	q_0	q_1	q_2
0	q_0	q_1	q_2
1	q_1	q_2	q_0

← transition table

$q_0 = q_0$
$F = \{q_0\}$.

def. Computation (given automata M, input $s = s_1 \dots s_n$)
sequence of states $r_0, \dots, r_n$ s.t.

→ $r_0 = q_0$
→ $r_i = \delta(r_{i-1}, s_i)$ for $i = 1 \dots n$.

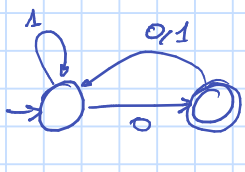
] in other words: $r_0 = q_0$
$r_1 = \delta(r_0, s_1)$
$r_2 = \delta(r_1, s_2)$
$r_3 = \delta(r_2, s_3)$
⋮
$r_n = \delta(r_{n-1}, s_n)$

Accepting if $r_n \in F$, rejecting o/w.

e.g. $(M_3, 010) \rightsquigarrow (q_0, q_0, q_1, q_1)$ (rej.).
$(M_3, 1101) \rightsquigarrow (q_0, q_1, q_2, q_2, q_0)$ (acc.).

def M recognizes language $\{s \in \Sigma^* \mid M \text{ accepts } s\}$.

e.g.



ε: ✗
0: ✓ 1: ✗
00: ✗ 01: ✗ 10: ✓ 11: ✗
000: ✓ 001: ✗ 010: ✓ 011: ✗
100: ✗ 101: ✗ 110: ✓ 111: ✗

Accepts strings of the form
"any amount of repetitions of either 00, 01, or 1, followed by a 0".

We'll see how to write in a more compact form.

Q. How to recognize complementary language?

A: Complement set of accepting states.

Q. How to recognize union of languages?

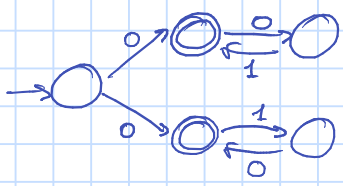
e.g. $L = \{0010101\dots\} \cup \{010101\dots\}$

We would like to run two automata in parallel, and accept iff one of them accepts.

This idea is called nondeterminism (because we are "guessing" which branch to take).

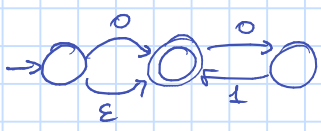
If we can guess at every transition, we get nondeterministic automata.

e.g.



Also convenient to allow branching on empty character

e.g.



Formally: like deterministic automata, but

$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to Q$

and not necessarily a function (one input can have 0, 1, or more outputs).

We write $((q, a), r) \in \delta$ or $r \in \delta(q, a)$ to mean r reachable from q on input a.

def computation: still the same, but not uniquely defined.

def M accepts s if $\exists s'$ st s' without $\varepsilon = s$ and $\exists$ accepting computation of (M, s').

Informal trick: use fingers / pebbles / ... to keep track of all possible states at every character in input.