

# Discrete Mathematics

**Non-determinism** — Spring 2025

Konstantin Chukharev

# §1 Non-determinism

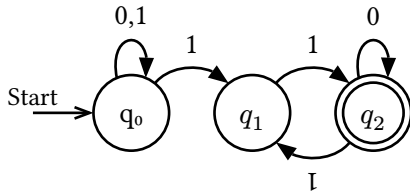
# Non-deterministic Finite Automata

**Definition 1:** A *non-deterministic finite automaton* (NFA) is a 5-tuple  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ , where

- $Q$  is a *finite* set of states,
- $\Sigma$  is an *alphabet* (finite set of input symbols),
- $\delta : Q \times \Sigma \longrightarrow \mathcal{P}(Q)$  is a *transition function*,
- $q_0 \in Q$  is an *initial* (*start*) state,
- $F \subseteq Q$  is a set of *accepting* (*final*) states.

**Note:**  $\delta : (q, c) \mapsto \underbrace{\{q^{(1)}, \dots, q^{(n)}\}}_{\text{non-determinism}}$

	0	1
q0	q0	q0, q1
q1		q2
q2	q2	q1



Michael Rabin



Dana Scott

# Non-Determinism

**Definition 2:** A model of computation is *deterministic* if at every point in the computation, there is exactly *one choice* that can make.

**Note:** The machine accepts if *that* series of choices leads to an accepting state.

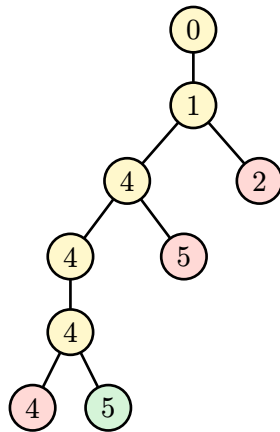
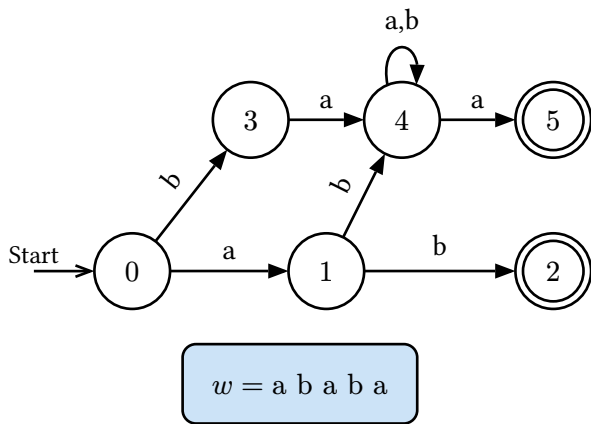
**Definition 3:** A model of computation is *non-deterministic* if the computing machine may have *multiple decisions* that it can make at one point.

**Note:** The machine accepts if *any* series of choices leads to an accepting state.

## **Intuition on non-determinism:**

1. Tree computation
2. Perfect guessing
3. Massive parallelism

## Tree Computation



- At each *decision point*, the automaton *clones* itself for each possible decision.
- The series of choices forms a directed, rooted *tree*.
- At the end, if *any* active accepting (*green*) states remain, we *accept*.

## Perfect Guessing

- We can view nondeterministic machines as having *magic superpowers* that enable them to *guess* the *correct choice* of moves to make.
- Machine can always guess the right choice if one exists.
- No physical implementation is known, yet.

## Massive Parallelism

- An NFA can be thought of as a DFA that can be in many states *at once*.
- Each symbol read causes a transition on every active state into each potential state that could be visited.
- Non-deterministic machines can be thought of as machines that can try any number of options in parallel (using an unlimited number of “processors”).

## Computation Model

Reachability relation for NFA is very similar to DFA's:

$$\langle q, x \rangle \vdash_{\text{DFA}} \langle r, y \rangle \quad \text{iff} \quad \begin{cases} x = cy & \text{where } c \in \Sigma \\ r = \delta(q, c) \end{cases}$$

$$\langle q, x \rangle \vdash_{\text{NFA}} \langle r, y \rangle \quad \text{iff} \quad \begin{cases} x = cy & \text{where } c \in \Sigma \\ r \in \delta(q, c) \end{cases}$$

**Definition 4:** An NFA *accepts* a word  $w \in \Sigma^*$  iff  $\langle q_0, w \rangle \vdash^* \langle f, \varepsilon \rangle$  for some  $f \in F$ .

**Definition 5:** A language *recognized* by an NFA is a set of all words accepted by the NFA.

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \langle q_0, w \rangle \vdash^* \langle f, \varepsilon \rangle, f \in F\}$$



## Rabin–Scott Powerset Construction

Any NFA can be converted to a DFA using *Rabin–Scott subset construction*.

$$\mathcal{A}_N = \langle \Sigma, Q_N, \delta_N, q_0, F_N \rangle$$

- $Q_N = \{q_1, q_2, \dots, q_n\}$
- $\delta_N : Q_N \times \Sigma \longrightarrow \mathcal{P}(Q_N)$

$$\mathcal{A}_D = \langle \Sigma, Q_D, \delta_D, \{q_0\}, F_D \rangle$$

- $Q_D = \mathcal{P}(Q_N) = \{\emptyset, \{q_1\}, \dots, \{q_2, q_4, q_5\}, \dots, Q_N\}$
- $\delta_D : Q_D \times \Sigma \longrightarrow Q_D$
- $\delta_D : (A, c) \mapsto \{r \mid \exists q \in A. r \in \delta_N(q, c)\}$
- $F_D = \{A \mid A \cap F_N \neq \emptyset\}$

## $\varepsilon$ -NFA

**Definition 6:** *Epsilon closure* of a state  $q$ , denoted  $E(q)$  or  $\varepsilon\text{-clo}(q)$ , is a set of states reachable from  $q$  by  $\varepsilon$ -transitions.

$$E(q) = \varepsilon\text{-clo}(q) = \left\{ r \in Q \mid \begin{array}{c} \textcircled{q} \xrightarrow{\varepsilon} \textcircled{r} \end{array} \right\}$$

This definition can be extended to the *sets of states*. For  $P \subseteq Q$ :

$$E(P) = \bigcup_{q \in P} E(q)$$

**Note:**  $q \in \varepsilon\text{-clo}(q)$  since each state has an *implicit*  $\varepsilon$ -loop.

*Example:* For the following NFA, epsilon closure of  $q$  is  $\varepsilon\text{-clo}(q) = \{q, r, s\}$ .



## From $\varepsilon$ -NFA to NFA

To construct NFA from  $\varepsilon$ -NFA:

1. Perform a transitive closure of  $\varepsilon$ -transitions.
  - After that, accepted words contain *no two consecutive*  $\varepsilon$ -transitions.
2. Back-propagate accepting states over  $\varepsilon$ -transitions.
  - After that, accepted words *do not end* with  $\varepsilon$ .
3. Perform symbol-transition back-closure over  $\varepsilon$ -transitions.
  - After that, accepted words *do not contain*  $\varepsilon$ -transitions.
4. Remove  $\varepsilon$ -transitions.
  - After that, you get an NFA.

# Kleene's Theorem

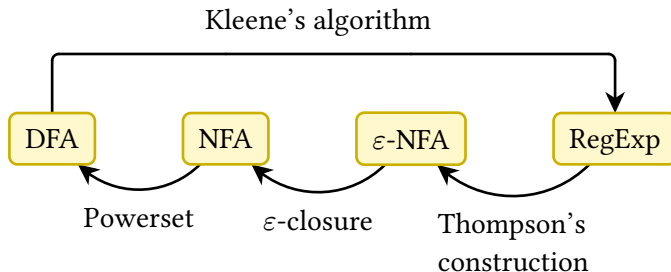
**Theorem 1:**  $\text{REG} = \text{AUT}$ .

**Proof** ( $\text{REG} \subseteq \text{AUT}$ ): *For every regular language, there is a DFA that recognizes it.*

Use *Thompson's construction* to build an  $\epsilon$ -NFA from regular expression, and then convert it to DFA. □

**Proof** ( $\text{AUT} \subseteq \text{REG}$ ): *The language recognized by a DFA is regular.*

Use *Kleene's algorithm* to construct a regular expression from an automaton. □

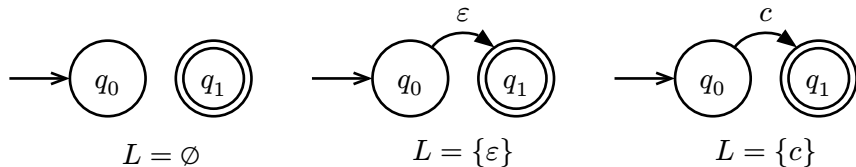


# Thompson's construction

**Definition 7:** *Thompson's construction* is a method of constructing an NFA from a regular expression.

Prove  $\text{REG} \subseteq \text{AUT}$  by induction over the *generation index*  $k$ . Show that  $\forall k. \text{Reg}_k \subseteq \text{AUT}$ .

**Base:**  $k = 0$ , construct automata for  $\text{Reg}_0 = \{\emptyset, \{\varepsilon\}, \{c\} \text{ for } c \in \Sigma\}$ , *showing*  $\text{Reg}_0 \subseteq \text{AUT}$ .



**Induction step:**  $k > 0$ , already have automata for languages  $L_1, L_2 \in \text{Reg}_{k-1}$ .

TODO: fancy pictures. For now, draw on the board.

## Kleene's Algorithm

**Definition 8:** *Kleene's algorithm* is a method of constructing a regular expression from a DFA.

Let  $R_{ij}^k$  be a set of all words that take  $\mathcal{A}$  from state  $q_i$  to  $q_j$  without going *through* (both entering and leaving) any state numbered higher than  $k$ . Note that  $i$  and  $j$  *can* be higher than  $k$ . Since all states are numbered 1 to  $n$ ,  $R_{ij}^n$  denotes the set of all words that take  $q_i$  to  $q_j$ . We can define  $R_{ij}^k$  recursively:

$$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}$$
$$R_{ij}^0 = \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & \text{if } i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\} & \text{if } i = j \end{cases}$$