NFA vs DFA

Definitions:

- **NFA (Nondeterministic Finite Automaton):** Allows for multiple transitions from a state for a given input symbol and ε-transitions (transitions without consuming an input symbol).

  $NFA = (Q, \mathit{\Sigma}, \mathit{\Delta}, q_0, F)$
  - Q: Set of states
  - $\mathit{\Sigma}$: Alphabet
  - $\mathit{\Delta}$: Transition function ($\mathit{\Delta} : Q \times \mathit{\Sigma}_\varepsilon \to P(Q)$) where $\mathit{\Sigma}_\varepsilon = \mathit{\Sigma} \cup \{\varepsilon\}$
  - q_0: Start state
  - F: Set of final states
- **DFA (Deterministic Finite Automaton):** Allows only one transition from a state for each input symbol.

$$DFA = (Q, \mathit{\Sigma}, \delta, q_0, F)$$

  - Q: Set of states
  - $\mathit{\Sigma}$: Alphabet
  - δ: Transition function ($\delta : Q \times \mathit{\Sigma} \to Q$)
  - q_0: Start state
  - F: Set of final states

Comparisons:

- **Expressive Power:** Both NFA and DFA recognize the same set of languages: the regular languages.
- **Transition Mechanism:** DFA has a unique transition for every symbol, while NFA can have multiple transitions or ε-transitions.
- **Equivalence:** Every NFA can be converted to an equivalent DFA using the

powerset construction, recognizing the same language.
- **Ease of Construction:** NFAs can be simpler and more intuitive to design for certain regular languages, whereas DFAs can be more verbose.

Regular and Non-Regular Languages:

- **Regular Language:** If a language can be recognized by some DFA or NFA, it is regular. Regular languages are described by regular expressions.
- **Non-Regular Language:** Languages that cannot be represented by any DFA or NFA. They require more computational power, such as a Pushdown Automaton (PDA) for context-free languages.

Examples:

- For a language over $\Sigma = \{0, 1\}$ where the third position from the end is 1, a DFA can be designed with states to keep track of the last three symbols read, and it will have a unique transition for every symbol in each state.