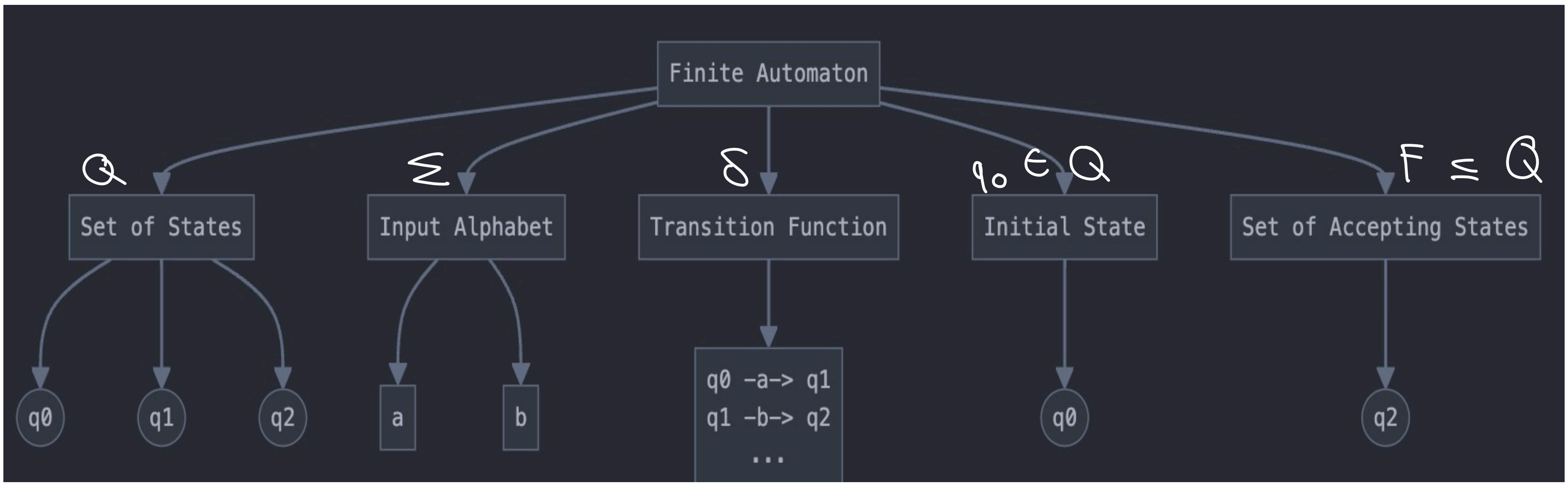
Finite Automata:

Finite automata (Finite State Machines) scan the characters, while keeping a finite set of states. When it sees a new character, it determines the next state to transition to. The next state could be another state or itself.



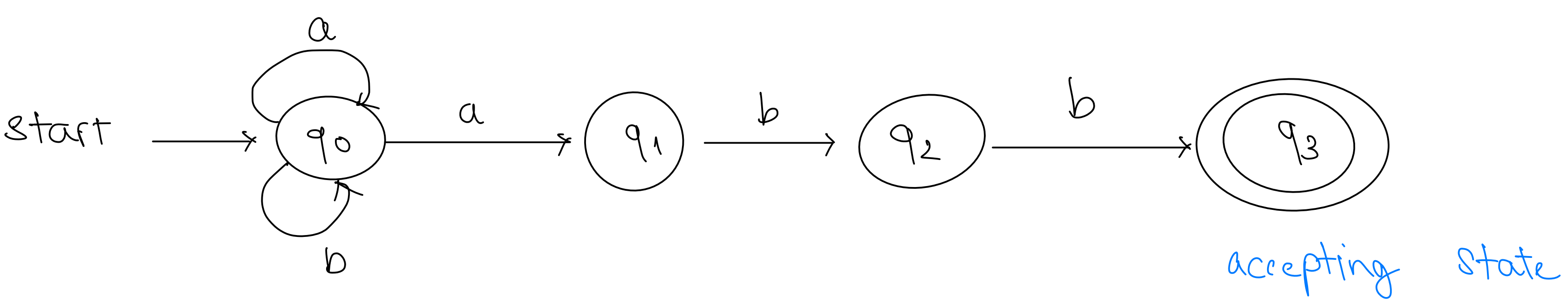
Types of finite automata:

- Deterministic Finite Automata (DFA):
  - For each state and input, there is exactly 1 next state
  - There is always a next state
  - Easier to implement

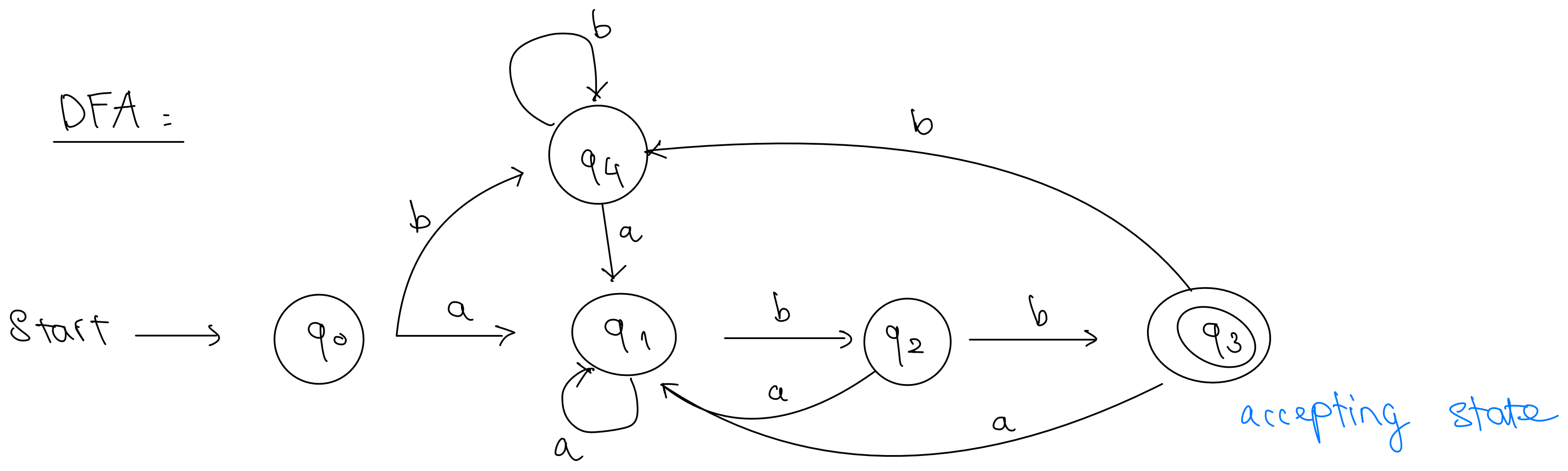
- Non-deterministic Finite Automata (NFA):
  - For each state and input, can have multiple next stages
  - May have ε-transition, which allow state change without input
  - Easier to design, harder to implement

Example: Show the NFA and DFA of this pattern
$(a \mid b)^* abb$

- NFA :



- DFA :



Maximal Munch Principle: (Principle of maximal scan)

It states that the lexer will try to match the longest possible string that form a valid token.

↳ The lexer doesn't immidiately output a token upon reaching an accepting state, it will keep going until no longer can extend the current token.

↳ If it can no longer match, it backtrack to the last accepting state.

Finite Automata is Used to implement Regular Languages

What is Regular Languages

Language is "regular" if it:

- Can be recognized by a finite automaton (finite state machine)
- Can be described using regular expression

Regular Languages Properties:

1. Closure properties:

Given languages R and S, the following languages are also regular:

- $R \cup S$
- $R \cap S$
- $R \setminus S = \{x \in \Sigma^* : x \in R \wedge x \notin S\}$ R without S
- $R \circ S = \{xy \in \Sigma^* : x \in R \wedge y \in S\}$ R concat S
- $\bar{R} = \{x \in \Sigma^* : x \notin R\}$ not R
- $R' = \{x \in \Sigma^* : x' \in R\}$ reverse R
- $R^i = \begin{cases} \{\varepsilon\}, & i = 0 \\ L^{i-1}L, & i > 0 \end{cases}$ multiple R's

- $R^* = \bigcup_{i=0}^{\infty} R^i = \{\varepsilon\} \cup R \cup RR \cup RRR \cup \dots$ Kleene star
- $R^+ = \bigcup_{i=1}^{\infty} R^i = R \cup RR \cup RRR \cup \dots$ Kleene plus

2. Pumping lemma property

"All sufficiently long string in regular language have a middle section that can be pumped any number of times and the resulting string would still be in the language."

↳ Why?

Because of finite automaton, which means finite states. If a string is too long, some states will be revisited.

⇒ Pumping lemma can be used to determine if a language is regular

Example:

Given $L = \{ 0^n 1^n \mid n \geq 0 \}$. Prove that L is non-regular.

- Assume L is regular
- Let n be pumping length given by the lemma
- Choose $s = 0^p 1^p$ (p zeros followed by p ones)
- According to lemma, s can be splitted into xyz, where $|xy| \leq n$ and $|y| \geq 1$ such that if we pump y by i times, $xy^iz \in L$.

But:

$\underset{x}{|}\underset{y}{00\dots}|\underset{z}{01\dots11}| \underset{\text{pump } y}{\Rightarrow} xy^iz \notin L$

⇒ Contradiction

⇒ L is not a regular language

3. Chomsky hierarchy:

Regular languages are the simplest class in Chomsky hierarchy