Kleene’s Theorem

A language $L$ is regular $\iff L = L(M)$ for some DFA $M$.

Recall: the notation $L(M)$ denotes the language of automata $M$.

A DFA $M = (\Sigma, Q, q_0, A, \delta)$.

We will show part of this proof over the next few lectures - CS 360 provides a complete proof.
Simulating a DFA

```c
int state = q0
char ch
while NOT EOF do
    read ch
    case state of
        q0:
            case ch of
                a: state = ...
                b: state = ...
                ...
        q1:
            case ch of
                ...
                ...
        qn:
            case ch of
                a: state = ...
                b: state = ...
                ...
    end case
end while
return state memberof A
```

DFAs with Actions

Typically, when working with finite automata in a theory class, we are mainly concerned with determining if a given input string is accepted or rejected.

In CS 241, we are using the finite automata to recognize the patterns for valid tokens (a Scanner); i.e. we want to recognize if the input would be accepted or ERROR and also have actions on the transitions.

- Compute the value of a number as we read it in digit by digit.
- Build a token character by character, emit the token.
Nondeterministic Finite Automata (NFA)

An NFA allows more than 1 transition on a given alphabet symbol from a state.

More complex? More powerful? Which path to take?
Formal Definition - NFA

Formally an NFA $M$ is a 5-tuple $M = (\Sigma, Q, q_0, A, \delta)$

- $\Sigma$ - non-empty, finite alphabet
- $Q$ - non-empty, finite set of states
- $q_0$ - start state
- $A \subseteq Q$ - set of accepting states
- $\delta : (Q \times \Sigma) \rightarrow \text{Subset of } Q$, $2^{|Q|}$ possible subsets.

Accept if there exists at least one path through $M$ that leads to an accepting state; reject if no such path exists.

Note: this doesn’t help us choose the correct path.
Acceptance in an NFA

Since the NFA transition function is different than a DFA, we must fix the extended transitions function and the definition of acceptance.

\[ \delta^* \text{ for NFAs: } (\text{Subset of states}) \times \Sigma^* \rightarrow (\text{Subset of states}) \]

**Base Case:**
\[ \delta^* (q_{\text{subset}}, \epsilon) = q_{\text{subset}} \]

**Recursive Case:**
\[
\delta^* (q_{\text{subset}}, cw) = \delta^* \left( \bigcup_{q \in q_{\text{subset}}} \delta(q, c), w \right)
\]

**Acceptance:** NFA \( M \) accepts \( w \) if \( \delta^* (\{q_0\}, w) \cap A \neq \emptyset \)
Simulating an NFA

states = \{q0\}
while NOT EOF do
    read ch
    states = Union(delta(q, ch) for each q in states)
end while
if states INTERSECT A != NULL
    Accept
else
    Reject
Conversion: NFA to a DFA

We will use the **Subset Construction Method**.

Idea: having read in part of the input string, we trace all paths that could be followed in the NFA; i.e. we have a set of states of the NFA that we could be in.

This “set of states" will be a single state in the DFA.

Label each DFA state as a “set of states" from the NFA.

**Note:** In CS 241, it is okay to skip drawing the state for \( \{\emptyset\} \) and the transitions that would lead there.
• The start state of the DFA is \( \{q_0\} \).

• For a state \( q_{\text{subset}} \) of the DFA and a given symbol \( c \) in the alphabet, \( c \in \Sigma \), determine the “set of states” in the NFA that can be reached from each \( q \in q_{\text{subset}} \) on \( c \). Create a new state in the DFA if this new “set of states” does not currently exist.

• Repeat the previous step until all DFA states have a transition on each alphabet symbol.

• Mark all states in the DFA as accepting if the state includes an accepting state from the NFA.