1 Finite automata

A **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states
- Σ is the finite alphabet
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states (final states)

A finite automaton reads an input cell-by-cell, transitioning between states depending on the current input symbol and state.

If A is the set of all strings accepted by a machine M, A is the **language** of M, denoted $L(M) = A$. If A is the language of M, M **recognises** A.

Can also say that M accepts A, but this is confusing – say "accept" when referring to machines accepting strings, and "recognise" when referring to machines accepting languages.

A machine only recognises one language, but can accept several strings. If a machine accepts no strings, it recognises the empty language $\emptyset$.

Question. Is zero accept states allowable?
Answer. Yes. Set F to be the empty set $\emptyset$, yielding zero accept states.

Question. Must there be exactly one transition exiting every state for each possible input symbol?
Answer. Yes. The transition function, δ, specifies one successor state for each possible combination of a state and an input symbol.

Finite automata are good models for computers with extremely limited memory.

1.1 Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1 w_2 ... w_n$ be a string, where each w_i is a member of alphabet Σ. M accepts w is there exists a sequence of states $r_0, r_1, ..., r_n \in Q$ where:

1. $r_0 = q_0$ (machine starts in the start state),
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, ..., n-1$ (machine goes between states according to the transition function), and
3. $r_n \in F$ (machine accepts an input if it ends in an accept state).

The machine M recognises language A if $A = \{w | M \text{ accepts } w\}$

2 Nondeterminism

In deterministic computation, when a machine is in a given state and reads the next input symbol, its next state is known (determined). In nondeterministic machines, there can be several choices for the next state.

Nondeterminism is just a generalisation of determinism. Every deterministic finite automaton (DFA) is automatically a nondeterministic finite automaton (NDFA).

Every state of a DFA has one exiting transition arrow for each symbol in the alphabet. In NDFAs, a state can have zero, one, or many exiting arrows for each symbol in the alphabet.

In DFAs, labels on transition arrows must be symbols from the alphabet. In NDFAs, labels on transition arrows may be symbols from the alphabet, or ϵ. There can be zero, one, or many arrows exiting each state with label ϵ.

2.1 How do nondeterministic machines compute?

Suppose a NDFA is running on an input string and gets to a state with multiple ways to proceed. After reading the input symbol, the machine splits into multiple copies of itself, and follows all possibilities in parallel. Each new machine takes one of the possibilities and continues.

If ϵ-labelled exiting arrows are encountered, the machine splits into multiple copies – one for each of the ϵ-labelled arrows and one staying at the current state – without reading an input.

In any of the new machines, if the next input symbol does not appear on any arrows exiting the current state, the machine (and the branch of computation associated with it) dies.

If any one of these machines is in an accept state at the end of the input, the input string is accepted by the NDFA.

Summary. Nondeterministic computation can viewed as a tree of possibilities, with the root being the start of the computation, and every branching point (fork) corresponds to a point where the machine has multiple choices.

If one of the branches ends in an accept state, the machine accepts.

Nondeterminism can be considered parallel computation wherein multiple independent "processes" or "threads" can be running concurrently.

2.2 Nondeterministic finite automata

DFAs and NDFAs differ in the type of transition function:

- In DFAs, the transition function takes a state and an input symbol, and produces the next state

- In NDFAs, the transition function takes a state and an input symbol or the empty string, and produces the set of possible next states

A **nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states
- Σ is the finite alphabet
- $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states (final states)

$\mathcal{P}(Q)$ is the power set (set of all subsets) of a set Q. For any alphabet Σ, write Σ_ϵ to mean $\Sigma \cup \{\epsilon\}$.

The **formal definition of computation** for a NDFA is also similar for that of a DFA. Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NDFA and w be a string over alphabet Σ. N accepts w if w can be written as $w = y_1 y_2 ... y_m$, where each y_i is a member of Σ_ϵ and there exists a sequence of states $r_0, r_1, ..., r_m \in Q$ where:

1. $r_0 = q_0$ (machine starts in the start state),
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, ..., m-1$ (state r_{i+1} is one of the allowable next states when N is in state r_i and reading y_{i+1}), and
3. $r_m \in F$ (machine accepts an input if it ends in an accept state).

Note that $\delta(r_i, y_{i+1})$ is the set of allowable next states, so say that r_{i+1} is a member of this set.

2.3 How are NDFAs useful?

- Every NDFA can be converted into an equivalent DFA – it is sometimes easier to construct NDFAs than directly constructing DFAs
- NDFAs can be much smaller than their DFA counterparts
- The functioning of some NDFAs is easier to understand than that of their DFA counterparts
- "Nondeterminism in finite automata is a good introduction to nondeterminism in more powerful computational models"

2.4 Equivalence

Two machines are **equivalent** if they recognise the same language.

Theorem. Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

If k is the number of states of the NDFA, it has 2^k subsets of states. Each subset corresponds to one of the possibilities that the DFA must remember, so there will be 2^k states in the DFA simulating the NDFA.

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NDFA recognising some language A. Construct DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognising A.

For any state R of M, define $E(R)$ to be the collection of states that can be reached from members of R only by going along ϵ arrows, including the members of R themselves. For $R \subseteq Q$ let:

$$E(R) = \{q | q \text{ can be reached from } R \text{ by travelling along 0 or more } \epsilon \text{ arrows}\}$$

The construction of M is as follows:

- $Q' = \mathcal{P}(Q)$ – every state of M is a set of states in N.
- For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q | q \in E(\delta(r, a))$ for some $r \in R\}$ – if R is a state of M, it is also a set of states in N. When M reads a symbol a in state R, it shows where a takes each state in R. As each state may go to a set of states, take the union of all these sets.
- $q_0' = E(\{q_0\})$ – M starts in the state corresponding to the set containing just the start state of N.
- $F' = \{R \in Q' | R$ contains an accept state of $N\}$ – M accepts if one of the possible states that N could be in at any point is an accept state. □

3 Regular languages

A language is a **regular language** if it is recognised by some finite automaton.

Theorem. A language is regular if and only if some nondeterministic finite automaton recognises it.

Proof.

($\rightarrow$)	As any NDFA can be converted into an equivalent DFA, if an NDFA recognises some languages, so does some DFA, and the language is regular.
($\leftarrow$)	A regular language has a DFA recognising it, and any DFA is also an NDFA.

3.1 Regular operations

Define three operations and language, called **regular operations**, to study properties of the regular languages.

Let A and B be languages.

- **Union:** $A \cup B = \{x | x \in A \text{ or } x \in B\}$
  Takes all strings in both A and B and puts them in one language.
- **Concatenation:** $A \circ B = \{xy | x \in A \text{ and } y \in B\}$
  Attaches a string from A in front of a string from B, in all possible ways, to get the strings in the new language.
- **Star:** $A^* = \{x_1 x_2 ... x_k | k \geq 0 \text{ and each } x_i \in A\}$
  A unary operation that attaches any number of strings in A together to get a string in the new language.
  As "any number" includes zero, ϵ is always a member of A^*.

3.2 Closure under operations

Theorem. The class of regular languages is closed under union.

Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognise A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognise A_2. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognise $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$
   The states of N are all the states of N_1 and N_2, with the addition of a new start state, q_0.
2. State q_0 is the start state of N
3. The set of accept states is $F = F_1 \cup F_2$
   N accepts if either N_1 or N_2 accepts, so the accept states of N are all those of N_1 and N_2.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon \end{cases}$$

Theorem. The class of regular languages is closed under concatenation.

Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognise A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognise A_2. Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognise $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$
2. The state q_1 is the same as the start state of N_1

3. Accept states F_2 are the same as the accept states of N_2.
   The accept states of N_1 have additional ϵ arrows that nondeterministically allow branching to N_2 whenever N_1 is in an accept state, signifying that N_1 has found an initial piece of the input that constitutes a string in A_1.
   N accepts when the input can be split into two parts – the first accepted by N_1 and the second by N_2.
   N nondeterministically "guesses" where to make the split.

4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ and } a = \epsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

Theorem. The class of regular languages is closed under star.

Given a regular language A_1, want to prove that A_1^* is also regular. Take a NDFA N_1 recognising A_1 and modify it to recognise A_1^*. The resulting NDFA N will accept its input whenever it can be broken into several pieces, with N_1 accepting each piece.

Construct N like N_1 with additional ϵ arrows returning to the start state from the accept states.
When processing gets to the end of a piece that N_1 accepts, N has the option of jumping back to the start state to try and read another piece that N_1 accepts. N must also accept ϵ as this is always a member of A_1^*.

Could add the start state to the set of accept states, but this may also add undesired strings to the recognised language, rather than just ϵ. Instead, add a new start state (which is also an accept state), and has an ϵ arrow to the old start state. This adds ϵ to the language without adding anything else.

Proof.