Topic 4 - Automata Theory

Week 7 - Automata Theory P1

basic definitions

subject	denotation	definition
alphabet	Σ	non-empty set of symbols
string/word		finite sequence of letters drawn from an alphabet
empty string	ϵ	strings with zero occurrences of letters; can be from any alphabet
length of a string, x	|x|	sum of occurrences of its symbols
set of all strings	Σ*	set of all strings composed from letters in Σ; like a powerset of a set
set of all non-empty strings	Σ+	set of all non-empty strings composed from letters in Σ
set of all strings of length *k*	Σ^k	set of all strings of length *k* composed from letters in Σ; size of Σ^k = |Σ|^k
language		collection of strings over an alphabet

computational model — idealised computer

finite state machine (or finite automaton) — simplest computational model

- formal definition: a finite automaton is a 5-tuple(consisting of 5 parts): set of states, input alphabet, rules for moving, start state, accept states
    - A finite automaton is a 5-tuple (Q, Σ, δ, q0, F), where
        1. **Q** is a finite set called the **states**,

2. **Σ** is a finites set called the **alphabet** (the set of your inputs, e.g {1,0}, {a, b}),
3. **δ: Q x Σ → Q** is the **transition function—**defines rules for moving
4. **q0 ∈ Q** is the **start state**
5. **F⊆Q** is the set of **accept states** (or **final states**)

- language of machine M — set of all strings that machine M accepts
    - denoted by: L(M) = A (where A is the set of all strings that machine M accepts)
    - we can say that **M recognises A** or **M accepts A** (but don't usually use *accepts* because different meaning where machines accepts strings and machines accept languages so use *recognises* instead)
- regular language — a language that some finite automaton recognises
    - regular operations — the three operations on languages
        - Let A and B be languages. We define the regular operations **union, concatenation, star** as follows:
            - union: A ∪ B = {x | x ∈ A or x ∈ B}
            - concatenation: A ∘ B = {xy | x ∈ A and y ∈ B}
                - attaches string in A in front of string in B in all possible ways to get the strings in the new language
            - star: A* = {x1, x2...xk | k ≥ 0 and each xi ∈ A}
                - attaches any number of strings in A together to get a string in the new language; 0 or an empty string ϵ is always included in A*
                - applies to single language rather than two different languages; it's a **unary operation** not a binary operation
        - example:

Let the alphabet Σ be the standard 26 letters $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$. If $A = \{\text{good}, \text{bad}\}$ and $B = \{\mathtt{boy}, \mathtt{girl}\}$, then

$A \cup B = \{\text{good}, \text{bad}, \text{boy}, \mathtt{girl}\}$,

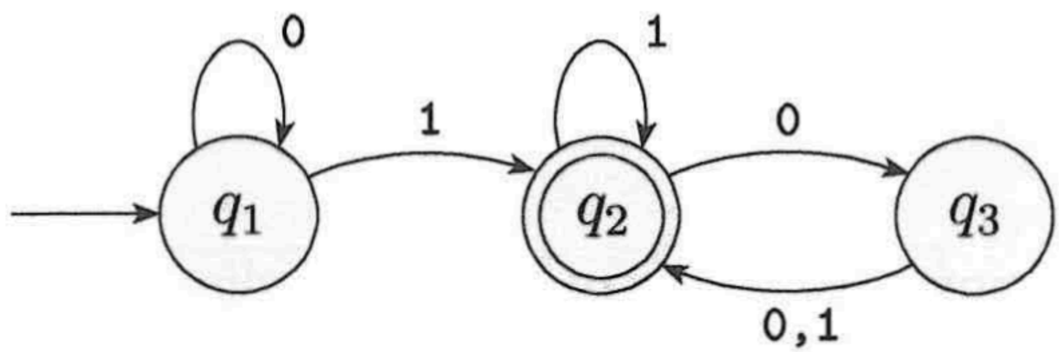
$A \circ B = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$, and

$A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badgood}, \text{badbad}, \text{goodgoodgood}, \text{goodgoodbad}, \text{goodbadgood}, \text{goodbadbad}, \ldots\}$.

        - a collection of objects is closed under some operation, if applying said operation to members of that collection returns an object still in that collection

    - to prove that a language is a regular one we have make a finite automaton, *M,* recognise said language: check proof on page 46 of FCS Automata

- good models for computers with very limited memory
- example of finite automaton M1:



    - above is the **state diagram** of M1
        - M1 has three **states,** q1,q2,q3
        - **starting state** — q1
        - **accept state** — q2; double circle; where your input must end for the automaton to output ACCEPT, otherwise output will be rejected
        - **transitions —** arrows pointing to the states
    - written in formal notation

- Q = {q1,q2,q3}
- Σ = {0,1}
- δ is described as

	0	1
q1	q1	q2
q2	q3	q2
q3	q2	q2

read as: if input to q1 is 0 then q1, if input to q1 is 1 then q2, etc

- q1 is the start state
- F = {q2}; q2 is the accepting/final state

- receives input from left to right, e.g 1011 is 1,0,1,1

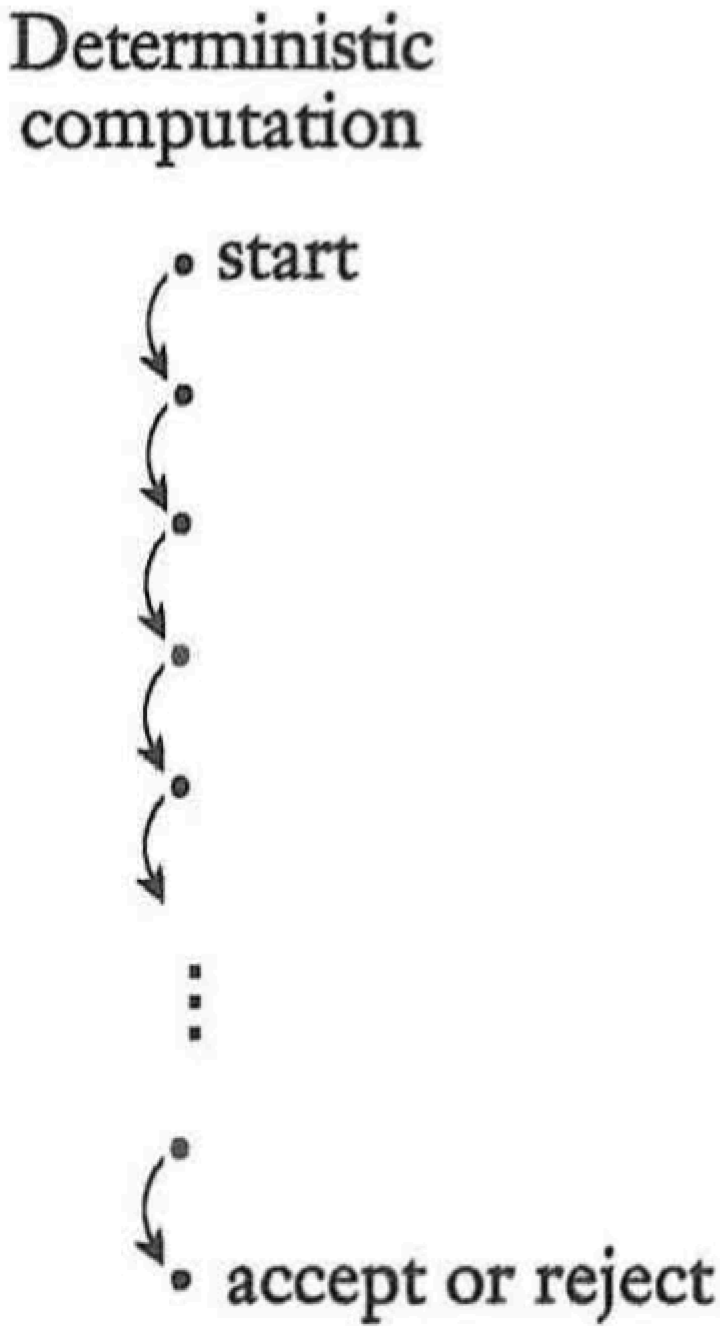
Week 8 - Automata Theory P2 (Deterministic and Nondeterministic Finite Automata)

Determinisim

deterministic computation — when we are able to determine the next state by reading the next input symbol of a given state

deterministic finite automata (DFA) — is a finite-state-machine that accepts or rejects a given string of symbols, by running through a state sequence uniquely determined by the string

- has exactly one exiting transition arrow for each symbol in the alphabet
- labels on the transition arrows are symbols from the alphabet
- simplest FA
- visual graph:



Nondeterminism

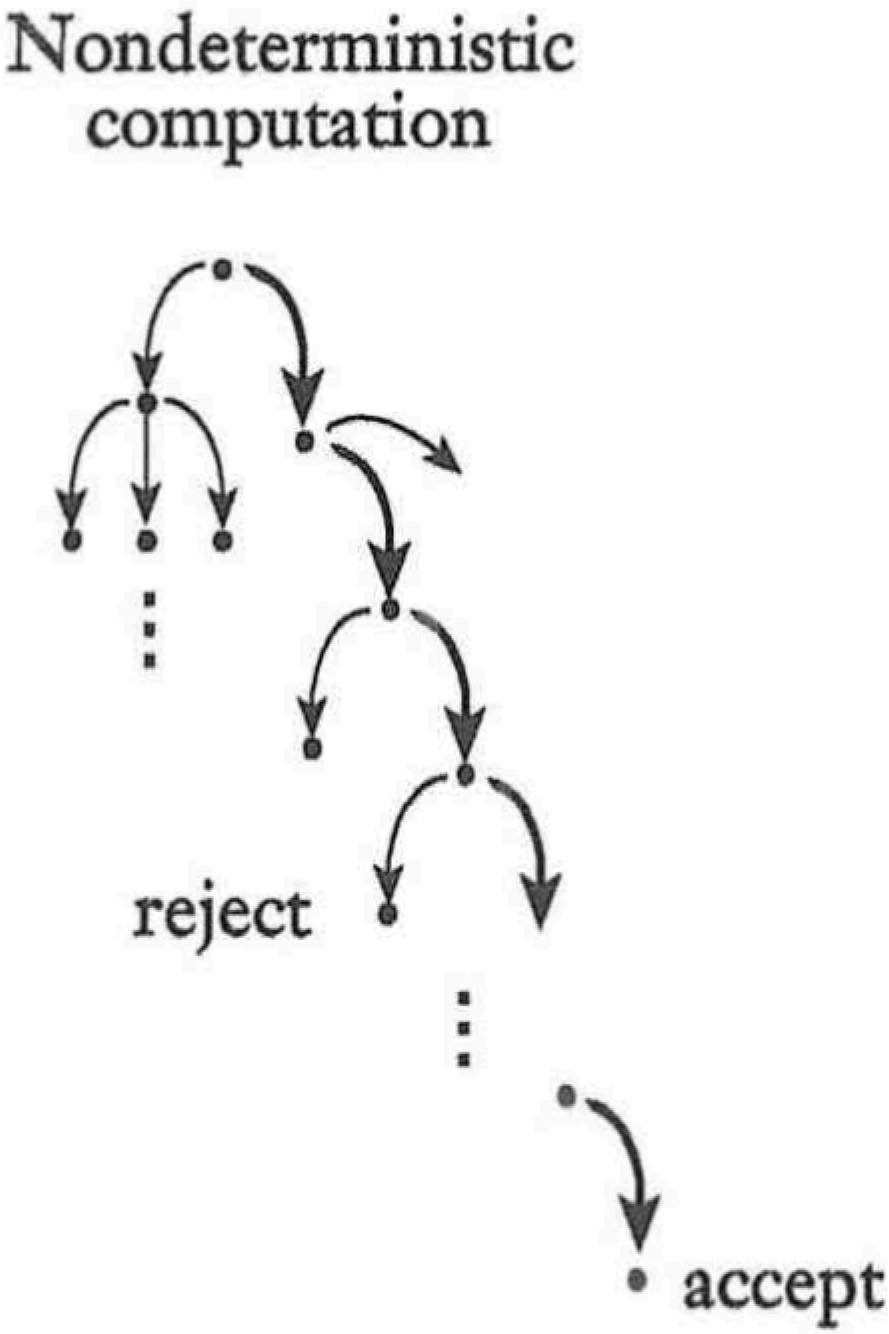
nondeterminism — a generalisation of determinisim

- kind of parallel computation wherein multiple independent “processes” or “threads” can running concurrently

nondeterministic finite automata (NFA) — a more general more of DFA and is not bound by the same laws as DFA

- formal definition: a nondeterministic finite automaton is a 5-tuple(consisting of 5 parts): set of states, input alphabet, rules for moving, start state, accept states
    - A nondeterministic finite automaton is a 5-tuple (Q, Σ, δ, q0, F), where
        1. **Q** is a finite set called the **states**,
        2. **Σ** is a finite **alphabet** (the set of your inputs, e.g {1,0}, {a, b}),
        3. **δ: Q x Σ → P(Q) (powerset of Q)** is the **transition function**—defines rules for moving
        4. **q0 ∈ Q** is the **start state**
        5. **F⊆Q** is the set of **accept states** (or **final states**)
    - similar to DFA except they differ in the type of transition function they have
        - DFA: transition function takes a state and an input symbol and produces the next state
        - NFA: transition function takes a state and an input symbol or the **empty string** and produces the **set of possible next states**
- every DFA is also a NFA
- a state may have zero, one, or many exiting arrows for each alphabet symbol
- may have arrows labeled with members of the alphabet or ϵ; zero, one, or many exiting arrows may have label of ϵ
- process of NFA:
    - machine splits into multiple copies and follows all possibilities in parallel once it encounters an input with several outputs
    - if next input symbol doesn't appear on any of the exiting arrows of a copy, then copy dies
    - if ANY ONE of the copies reach the accept state, then NFA accepts the input
- process of NFA with ϵ exiting symbols:
    - splits into multiple copies, with one staying at current state

    - then proceeds nondeterministically
- when NFA splits it **forks** into several children
- visual graph:



Equivalence of NFAs and DFAs

- recognise same class of languages
- every NFA has an equivalent DFA