

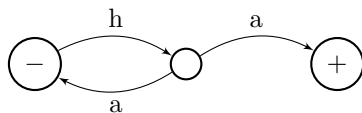
1 Finite State Automata

1.1 Alphabets & Strings

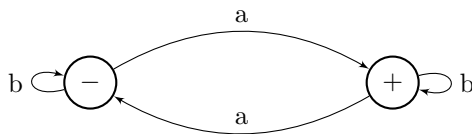
- Let A be a set; then A^n is the set of all finite sequences $a_1 \dots a_n$ with $a_i \in A$, $1 \leq i \leq n$
 - Elements of A are *letters* or *symbols*
 - Elements of A^n are *words* or *strings* over A of length n
- ε is the special *empty string*, the only string of length 0
- $A^+ = \bigcup_{m \geq 1} A^m$ – the set of non-empty strings over A of any length
- $A^* = A^+ \cup \varepsilon = \bigcup_{m \geq 0} A^m$ – the set of (possibly empty) strings over A of any length
- If $\alpha = a_1 \dots a_m$, $\beta = b_1 \dots b_n \in A^*$, then define $\alpha\beta$ to be $a_1 \dots a_m b_1 \dots b_n \in A^{m+n}$. This gives binary ‘product’ or *concatenation* on A^*
- For $\alpha \in A^+$, define $\alpha^n, n \in \mathbb{N}$ by $\alpha^0 = \varepsilon$, and $\alpha^{n+1} = \alpha^n \alpha$
- A *language* with alphabet A is a subset of A^*

1.2 Definition of an FSA

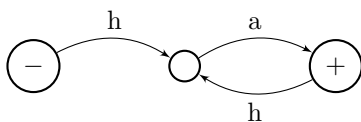
- A Finite State Automaton (FSA) is a tuple $M = (Q, F, A, \tau, q_0)$
 - Q is a finite set of states
 - $F \subseteq Q$ is the set of final states
 - A is the alphabet
 - $\tau \subseteq Q \times A \times Q$ is the set of transitions
 - $q_0 \in Q$ is the initial state
- The transition diagram of an FSA is a directed graph with:
 - Vertex set Q
 - An edge for each transition; $(q, a, q') \in \tau$ corresponds to an edge from q to q' with label a
 - Initial state q_0 labelled with $-$
 - Final states labelled with $+$
 - Example: a *non-deterministic* ‘haha machine’, with $A = \{h, a\}$



- A *computation* of M is a sequence $q_0, a_1, q_1, a_2, \dots, a_n, q_n$ with $n \geq 0$ where $(q_i, a_{i+1}, q_{i+1}) \in \tau$ for $0 \leq i \leq n-1$
 - The *label* on the computation is $a_1 \dots a_n$
 - The computation is *successful* if $q_n \in F$
 - A string $a_1 \dots a_n$ is *accepted* by M if there is a successful computation with label $a_1 \dots a_n$, and it is *rejected* otherwise
- The language recognised by M is $\mathcal{L}(M) = \{w \in A^* \mid w \text{ is accepted by } M\}$
- There is a one-to-one correspondence between computations of M and paths in the graph from q_0
- Example: $A = \{a, b\}$ of an FSA accepting only words with an odd number of ‘a’s



- An FSA is deterministic (a DFA) if for all $q \in Q, a \in A$ there is exactly one $q' \in Q$ such that $(q, a, q') \in \tau$
- Example: DFA for the ‘haha machine’



- Note this machine lacks a transition for a when in the initial state – though technically required for a DFA, it is easily fixed by adding an ‘error state’ to catch what would otherwise be missing transitions

1.3 Deterministic FSAs

- For a DFA M , define the transition function $\delta : Q \times A \rightarrow Q$ by $q' = \delta(a, q)$, where q' is the unique element such that $(q, a, q') \in \tau$
- If \mathcal{L} is a language with alphabet A , then the following are equivalent:
 1. \mathcal{L} is recognised by an FSA
 2. \mathcal{L} is recognised by a DFA
- Given a non-deterministic FSA $M = (Q, F, A, \tau, q_0)$, an equivalent DFA $M' = (Q', F', A, \tau', q'_0)$ may be generated by the *powerset method*:
 - $Q' = \mathcal{P}(Q) \setminus \emptyset$ (i.e. the set of all subsets of Q that aren't empty)
 - $F' = \{X \in Q' \mid q \in X \text{ for some } q \in F\}$
 - For $X \in Q', a \in A$, define $\delta(X, a) := \{q \in Q \mid (x, a, q) \in \tau \text{ for some } x \in X\}$
 - $\tau' = \{(X, a, \delta(X, a)) \mid X \in Q', a \in A\}$
 - $q'_0 = \{q_0\}$
- Proof: show that $\mathcal{L}(M) = \mathcal{L}(M')$
 - $\mathcal{L}(M) \subseteq \text{Lang}(M')$:
 - * Given $w \in \mathcal{L}(M)$, $q_0 a_1 \dots a_n q_n$ is a successful computation of M
 - * Then define $q'_i = \delta(q'_{i-1}, a_i)$ for $1 \leq i \leq n$
 - * $q'_0, a_1, q'_1 \dots a_n, q'_n$ will be a successful computation of M'
 - * Therefore $w \in \mathcal{L}(M')$
 - $\mathcal{L}(M') \subseteq \text{Lang}(M)$:
 - * Let $w = a_1 \dots a_n \in L(M')$, and $q'_0, a_1, q'_1 \dots a_n, q'_n$ be a successful computation of M'
 - * Each q'_i cannot be the empty set
 - * By definition of τ' , $\exists q_1 \in q'_1$ s.t. $(q_0, a_1, q_1) \in \tau$
 - * Then we can find $q_i \in q'_i$ s.t. $(q_{i-1}, a_i, q_i) \in \tau$ for $1 \leq i \leq n$
 - * For q_n we further require $q_n \in F$
 - * Therefore, $q_0, a_1, q_1, a_2, \dots, a_n, q_n$ is a successful computation
 - * Therefore $w \in \mathcal{L}(M)$

1.4 The Pumping Lemma

- The Pumping Lemma says that for any \mathcal{L} recognised by an FSA M , there is a certain word length beyond which all words can be split into sections as xyz , where xy^nz is also in the language
- Formally there is an integer $p > 0$ s.t. any word $w \in L$ with $|w| \geq p$ is of the form $w = xyz$, where $|y| > 0$, $|xy| \leq p$ and $xy^iz \in \mathcal{L}$ for $i \geq 0$
- Proof:
 - Let p be the number of states in M , and suppose $w = a_1 \dots a_n \in \mathcal{L}$, where $n \geq p$
 - A successful computation q_0, a_1, \dots, q_n has to pass through a certain state at least twice (by the pigeonhole principle)
 - Therefore, $\exists r < s$ s.t. $q_r = q_s$; choose minimal such s
 - Now put $x = a_1 \dots a_r$, $y = a_{r+1} \dots a_s$ (note $|y| > 0$), and $z = a_{s+1} \dots a_n$
 - By minimality of s , q_0, \dots, q_{s-1} are distinct, and $|xy| = s \leq p$
 - Then, note that q_r, a_{r+1}, \dots, q_s is a loop, which may be validly repeated $i \geq 0$ times
 - Therefore, $xy^iz \in \mathcal{L}$
- Corollary: here exist languages which are not computable by an FSA
- Example: there is no FSA which can recognise $\mathcal{L} = \{a^n b^n \mid n \in \mathbb{N}\}$
- Proof:
 - Assume for a contradiction there exists an FSA M which can recognise \mathcal{L}
 - Let p be the number from the pumping lemma, and choose $n \geq p$ and consider $w = a^n b^n$
 - By the pumping lemma, $\exists x, y, z$ s.t. $a^n b^n = xyz$, with $|y| \geq 1$ and $|xy| \leq p \leq n$
 - Then y is written entirely in terms of the letter a , and $|y| \geq 1$
 - By the pumping lemma, $xy^iz \in \mathcal{L}$ for all i
 - So choose $i = 0$, then some $w = a^k b^n \in \mathcal{L}$ s.t. $k < n$, which is a contradiction

2 Turing Machines

2.1 Definition

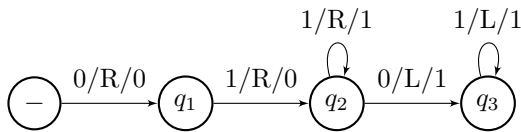
- A Turing machine is a tuple $T = (Q, F, A, I, \tau, q_0)$
 - Q is a finite set of states
 - $F \subseteq Q$ is the set of final states
 - A is a finite set, the tape alphabet, with a distinguished blank symbol $B \in A$
 - I is a subset of $A \setminus \{B\}$, the input alphabet
 - $\tau \subseteq Q \times A \times Q \times A \times \{L, R\}$ is the set of transitions
 - $q_0 \in Q$ is the initial state
- As in an FSA, non-determinism is allowed
- The tape is infinite in both directions, but only ever contains a finite number of non-blank symbols
- A *tape description* for T is a triple (a, α, β) with $a \in A$, and $\alpha : \mathbb{N} \rightarrow A$ and $\beta : \mathbb{N} \rightarrow A$ being functions with $a(n) = B$ and $\beta(n) = B$ for all but finitely many $n \in \mathbb{N}$
 - So the tape looks like: $\dots BBB\beta(l)\beta(l-1)\dots\beta(0)\underline{a}(0)\alpha(1)\dots\alpha(r)BBB\dots$, with $l, r \in \mathbb{N}$
- A *configuration* of T is a tuple (q, a, α, β) where $q \in Q$ and (a, α, β) is a tape description
- If $c = (q, a, \alpha, \beta)$ is a configuration, a configuration c' is obtained (reachable) from c by a single move if one of the following holds:
 - $(q, a, q', a', L) \in \tau$ and $c' = (q', \beta(0), \alpha', \beta')$ where: $\alpha'(0) = a', \alpha'(n) = \alpha(n-1), n > 0$ and $\beta'(n) = \beta(n+1), n \geq 0$, or
 - $(q, a, q', a', R) \in \tau$ and $c' = (q', \alpha(0), \alpha', \beta')$ where: $\alpha'(n) = \alpha(n+1), n \geq 0$ and $\beta'(0) = a', \beta'(n) = \beta(n-1), n > 0$
- A *computation* of T is a finite sequence of configurations $c_1, \dots, c_n = c'$ where $n \geq 1$ and c_{i+1} is obtained from c_i by a single move, for $1 \leq i \leq n-1$
- A configuration is *terminal* if no configuration is reachable from it
- A computation halts if c' is terminal (i.e. there is no configuration reachable from c')
- We may write $c \xrightarrow{T} c'$ if there is a computation starting at c and ending at c'

2.2 Turing Machine as Language Recogniser

- For $w = a_1 \dots a_n \in A^*$, let $c_w = (a_0, \underline{a_1} \dots a_n)$ (recall $\underline{a_1} \dots a_n$ is a tape description (a, α, β))
- If $w = \varepsilon$, we put $c_w = (q_0, \underline{B})$
- The TM T *accepts* if $c_w \xrightarrow{T} c'$ for some $c' = (q, a, \alpha, \beta)$ with $q \in F$
- The language recognised by T is $\mathcal{L}(T) = \{w \in I^* \mid w \text{ is accepted by } T\}$
- Note that $\mathcal{L}(T)$ is a language over I rather than over A
- T is deterministic if for every $(q, a) \in Q \times A$ there is *at most one* element of τ starting with (q, a)
- Then, there is at most one config c' obtained from c by a single move; set $\delta(c) = c'$
- $\delta : C \rightarrow C$ is then a partial function

2.3 Numerical Turing Machines: TMs as Function Calculators

- We want to use TMs to describe a partial function $f : \mathbb{N}^n \rightarrow \mathbb{N}$
- A *numerical TM* is a deterministic TM $T = (Q, F, A, I, \tau, q_0)$ with:
 - $F = I = \emptyset$
 - $A = \{0, 1\}$, with 0 as the blank symbol
- In a numerical TM, the final states F and input alphabets I are not relevant
- For $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{N}^n$, define the tape description $Tape(\mathbf{x}) = \underline{0}1^{x_1}\underline{0}1^{x_2}\underline{0} \dots \underline{0}1^{x_n}$
- Define the partial function $\varphi_{T,n} : \mathbb{N}^n \rightarrow \mathbb{N}$ as follows:
 - Let $\mathbf{x} \in \mathbb{N}^n$ be given
 - The initial config of T is $(q_0, Tape(\mathbf{x}))$
 - If T halts with tape $\underline{0}1^y = Tape(y)$ for some $y \in \mathbb{N}$, then $\varphi_{T,n}(\mathbf{x}) = y$
 - Otherwise, $\varphi_{T,n}$ is undefined
- If $f : \mathbb{N}^n \rightarrow \mathbb{N} = \varphi_{T,n}$ for some numerical TM T , then f is *TM computable*
- Note that when considering TMs as language recognisers, halting is regarded as an error – but for a numerical TM, it is fine *so long as* it ends with a configuration of the form $(q, \underline{0}1^y)$ with $y \in \mathbb{N}$
- Example: an addition function $S : \mathbb{N}^2 \rightarrow \mathbb{N}$



- Ultimate theorem: All TM computable functions are partial recursive, and conversely all partial recursive functions are TM computable

3 Partial Recursive Functions

3.1 Partial Functions, Definition by Composition & Primitive Recursion

- Classes of functions:
 - Let P be the set of partial functions, $P = \{f \mid f \text{ is a partial function } \mathbb{N}^n \rightarrow \mathbb{N} \text{ for some } n > 0\}$
 - Let T be the set of total functions, $T = \{f \in P \mid f \text{ is total}\}$
 - A *class* of functions means a subset of P , and a class of total functions means a subset of T
 - Goal: build a class of functions which we might call ‘computable’

- Let $g : \mathbb{N}^r \rightarrow \mathbb{N}, h_1, \dots, h_r : \mathbb{N}^n \rightarrow \mathbb{N}$ be partial functions.

Then the partial function $f : \mathbb{N}^n \rightarrow \mathbb{N}$ obtained from g, h_1, \dots, h_r by composition is defined by:

$$f(\mathbf{x}) = g(h_1(\mathbf{x}), \dots, h_r(\mathbf{x}))$$

- We write $f = g \circ (h_1, \dots, h_r)$

- Let $g : \mathbb{N}^n \rightarrow \mathbb{N}, h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ be partial functions.

Then the partial function $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ obtained from g and h by primitive recursion is defined by:

$$f(\mathbf{x}, 0) = g(\mathbf{x})$$

$$f(\mathbf{x}, y + 1) = h(\mathbf{x}, y, f(\mathbf{x}, y))$$

- For a given \mathbf{x} , $f(\mathbf{x}, y)$ is defined for no y , for all y , or for $0 \leq y \leq r$ for some $r \in \mathbb{N}$
- Where the ‘counter’ parameter is placed does not matter - it could equally be at the start

3.2 Partial Recursive Functions

- We define the *initial functions* to be the following functions:
 - The zero function $z : \mathbb{N} \rightarrow \mathbb{N}$, such that $z(x) = 0$ for all $x \in \mathbb{N}$
 - The successor function $\sigma : \mathbb{N} \rightarrow \mathbb{N}$, such that $\sigma(x) = x + 1$ for all $x \in \mathbb{N}$
 - The projection functions $\pi_{i,n} : \mathbb{N}^n \rightarrow \mathbb{N}$, where for $n \geq 1$ and $1 \leq i \leq n$, $\pi_{i,n}(x_1, \dots, x_n) = x_i$
- A class \mathcal{C} of total functions is *primitively recursively closed* if:
 - \mathcal{C} contains all the initial functions
 - \mathcal{C} is closed under composition
 - \mathcal{C} is closed under primitive recursion
- The smallest primitively recursively closed class (i.e. the intersection of all prim. rec. closed classes) is called *the class of primitive recursive functions*
- Example: addition function $S : \mathbb{N}^2 \rightarrow \mathbb{N}$, such that $S(x, y) = x + y$

$$\begin{aligned} S(x, 0) &= g(x), g = \pi_{1,1} \\ S(x, y + 1) &= S(x, y) + 1 \\ &= \sigma(S(x, y)) \\ &= h(x, y, S(x, y)), h = \sigma \circ \pi_{3,3} \end{aligned}$$

4 First Order Logic