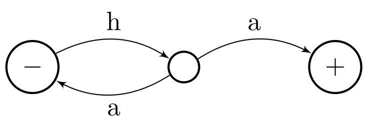
1 Finite State Automata

1.1 Alphabets & Strings

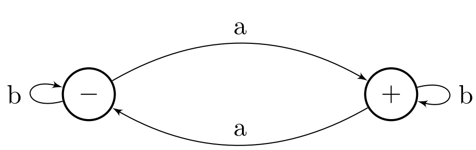
- Let A be a set; then A^n is the set of all finite sequences $a_1 \dots a_n$ with $a_i \in A$, $1 \leq i \leq m$
  - Elements of A are *letters* or *symbols*
  - Elements of A^n are *words* or *strings* over A of length m
- ε is the special *empty string*, the only string of length 0
- $A^+ = \bigcup_{m \geq 1} A^m$ – 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 $\alpha = a_1 \dots a_m$, $\beta = b_1 \dots b_m \in A^*$, then define $\alpha\beta$ to be $a_1 \dots a_m b_1 \dots b_m \in A^{m+n}$. 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

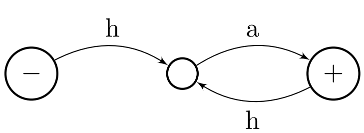
- A Finite State Automaton (FSA) is a tuple $M = (Q, F, A, \tau, q_0)$
  - Q is a finite set of states
  - $F \subseteq Q$ is the set of final states
  - A is the alphabet
  - $\tau \subseteq Q \times A \times Q$ is the set of transitions
  - $q_0 \in Q$ is the initial state
- The transition diagram 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 $q_0, a_1, q_1, a_2, \ldots, a_n, q_n$ 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 $a_1 \ldots a_m$
  - The computation is *successful* if $q_n \in F$
  - A string $a_1 \ldots a_n$ is *accepted* by M if there is a successful computation with label $a_1 \ldots a_n$, 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 $\delta : Q \times A \to Q$ 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)
  - $F' = \{X \in Q' \,|\, q \in X \text{ for some } q \in F\}$
  - For $X \in Q', a \in A$, define $\delta(X, a) := \{q \in Q \,|\, (x, a, q) \in \tau \text{ for some } x \in X\}$
  - $\tau' = \{(X, a, \delta(X, a)) \,|\, X \in Q', a \in A\}$
  - $q_0' = \{q_0\}$
- Proof: show that $\mathcal{L}(M) = \mathcal{L}(M')$
  - $\mathcal{L}(M) \subseteq Lang(M')$:
    * 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 $w \in \mathcal{L}(M')$
  - $\mathcal{L}(M') \subseteq Lang(M)$:
    * 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 $\mathcal{L} = \{a^nb^n \mid n \in \mathbb{N}\}$
- 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 $w = a^nb^n$
  - By the pumping lemma, $\exists x, y, z$ s.t. $a^nb^n = xyz$, 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 $w = a^kb^n \in \mathcal{L}$ s.t. $k < n$, which is a contradiction

2 Turing Machines

2.1 Definition

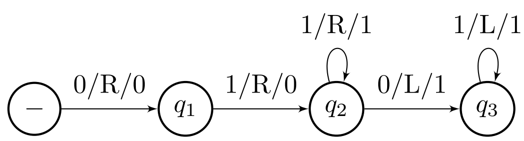
- A Turing machine is a tuple $T = (Q, F, A, I, \tau, q_0)$
  - Q is a finite set of states
  - $F \subseteq Q$ is the set of final states
  - A is a finite set, the tape alphabet, with a distinguished blank symbol $B \in A$
  - I is a subset of $A \setminus \{B\}$, the input alphabet
  - $\tau \subseteq Q \times A \times Q \times A \times \{L, R\}$ 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 $\alpha : \mathbb{N} \to A$ and $\beta : \mathbb{N} \to A$ being functions with $a(n) = B$ and $B(n) = B$ for all but finitely many $n \in \mathbb{N}$
  - So the tape looks like: $\ldots BBB\beta(l)\beta(l-1)\ldots\beta(0)\underline{a}\alpha(0)\alpha(1)\ldots\alpha(r)BBB\ldots$, 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 $c_1, \ldots, c_n = c'$ 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 $c \underset{T}{\to} c'$ if there is a computation starting at c and ending at c'

2.2 Turing Machine as Language Recogniser

- For $w = a_1 \ldots a_n \in A^*$, let $c_w = (a_0, \underline{a_1} \ldots a_n)$ (recall $\underline{a_1} \ldots a_n$ is a tape description (a, α, β))
- If $w = \varepsilon$, we put $c_w = (q_0, \underline{B})$
- The TM T *accepts* if $c_w \underset{T}{\to} c'$ 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 $\delta(c) = c'$
- $\delta : C \to C$ is then a partial function

2.3 Numerical Turing Machines: TMs as Function Calculators

- We want to use TMs to describe a partial function $f : \mathbb{N}^n \to \mathbb{N}$
- 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 $Tape(\mathbf{x}) = \underline{0}1^{x_1}01^{x_2}0 \dots 01^{x_n}$
- Define the partial function $\varphi_{T,n} : \mathbb{N}^n \to \mathbb{N}$ 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 $f : \mathbb{N}^n \to \mathbb{N} = \varphi_{T,n}$ 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 $S : \mathbb{N}^2 \to \mathbb{N}$



- 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, $P = \{f \mid f \text{ is a partial function } \mathbb{N}^n \to \mathbb{N} \text{ for some } n > 0\}$
  - 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 $g : \mathbb{N}^r \to \mathbb{N}, h_1 \ldots h_r : \mathbb{N}^n \to \mathbb{N}$ be partial functions.

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

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

  - We write $f = g \circ (h_1, \ldots, h_r)$
- Let $g : \mathbb{N}^n \to \mathbb{N}, h : \mathbb{N}^{n+1} \to \mathbb{N}$ be partial functions.

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

  $$\begin{aligned} f(\mathbf{x}, 0) &= g(\mathbf{x}) \\ f(\mathbf{x}, y+1) &= h(\mathbf{x}, y, f(\mathbf{x}, y)) \end{aligned}$$

  - 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 Primitive Recursive Functions

- We define the *initial functions* to be the following functions:
  - The zero function $z : \mathbb{N} \to \mathbb{N}$, such that $z(x) = 0$ for all $x \in \mathbb{N}$
  - The successor function $\sigma : \mathbb{N} \to \mathbb{N}$, such that $\sigma(x) = x + 1$ for all $x \in \mathbb{N}$
  - The projection functions $\pi_{i,n} : \mathbb{N}^n \to \mathbb{N}$, where for $n \geq 1$ and $1 \leq i \leq n$, $\pi_{i,n}(x_1, \ldots, x_n) = x_i$
- 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 $S : \mathbb{N}^2 \to \mathbb{N}$, such that $S(x, y) = x + y$

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

- Useful tips for showing a function is in a primitively recursively closed class $\mathcal{C}$:
  - Given $f : \mathbb{N}^n \to \mathbb{N}$ is in $\mathcal{C}$

    If $g : \mathbb{N}^m \to \mathbb{N}$ is defined by $g(x_1, \ldots, x_m) = f(y_1, \ldots, y_n)$ where each y_i is either a constant or x_j for some j, then $g \in \mathcal{C}$ – lets you manipulate arity
  - To show a unary function $f : \mathbb{N} \to \mathbb{N}$ is in $\mathcal{C}$ by primitive recursion, define $f' : \mathbb{N}^2 \to \mathbb{N}$ such that $f'(x, y) = f'(y)$; then, if f' can be shown to be in $\mathcal{C}$, f will be also
  - Let $a \in \mathbb{N}$ and $h : \mathbb{N} \to \mathbb{N}$ be in $\mathcal{C}$

    Then, for $f : \mathbb{N} \to \mathbb{N}$, if $f(0) = a$ and $f(y + 1) = h(f(y))$, $f \in \mathcal{C}$
- A *primitive recursive definition* of $f : \mathbb{N}^n \to \mathbb{N}$ is a finite sequence of functions $f_0, f_1, \ldots, f_k = f$, where for each i:
  - f_i is initial; or
  - f_i is obtained from composition of some functions f_j, $j < i$; or
  - f_i is obtained by primitive recursion from two of f_j, $j < i$
- Example: addition function S can be defined by $\pi_{1,1}, \pi_{3,3}, \sigma, \sigma \circ \pi_{3,3}$
- The class $\mathcal{C}_1$ of primitive recursive functions is the same as the class $\mathcal{C}_2$ of functions that have a primitive recursive definition (seems trivial, but isn't!)

  Prove by showing $\mathcal{C}_1 \subseteq \mathcal{C}_2$ (i.e. $\mathcal{C}_2$ is prim. rec. closed) and that $\mathcal{C}_2 \subseteq \mathcal{C}_1$ (i.e. $\mathcal{C}_2$ is contained in any prim. rec. closed class)
- Let $\mathcal{C}$ be a prim. rec. closed class, and let $g : \mathbb{N}^{n+1} \to \mathbb{N}$ be in $\mathcal{C}$; then the functions $f_1 : \mathbb{N}^{n+1} \to \mathbb{N}$ and $f_2 : \mathbb{N}^{n+1} \to \mathbb{N}$ defined by:

  $f_1(\mathbf{x}, y) = \sum_{t=0}^{y} g(\mathbf{x}, t)$

  $f_2(\mathbf{x}, y) = \prod_{t=0}^{y} g(\mathbf{x}, t)$

  are also in $\mathcal{C}$
- Useful prim. rec. functions:
  - Proper subtraction $x \dot{-} y = max\,\{x - y, 0\}$
  - Sign $sg(x) = \begin{cases} 0 \text{ if } x = 0 \\ 1 \text{ if } x \geq 0 \end{cases}$

3.3 Predicates

- A predicate $P(x_1, \ldots, x_n)$ of n variables is a statement concerning $x_i \in \mathbb{N}$ which is either true or false
- We can identify P with the set $A_P = \{\mathbf{x} \in \mathbb{N}^n \mid P(\mathbf{x}) \text{ is true}\}$

  E.g. $P(x, y)$ means "x divides y", so $A_P = \{(1,6), (2,6), (3,6), (6,6), (1,3) \ldots\}$
- The *characteristic function* of a set $\chi_A : \mathbb{N}^n \to \{0,1\}$ of $A \subseteq \mathbb{N}^n$ is defined by:

$$\chi_A(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{x} \in A \\ 0 \text{ if } \mathbf{x} \notin A \end{cases}$$

- For a predicate P, we define χ_P to be χ_{A_P}
- Let $\mathcal{C}$ be a prim. rec. closed class; then a subset $A \subseteq \mathbb{N}^n$ is in $\mathcal{C}$ if $\chi_A \in \mathcal{C}$

  So a predicate P of n variables is in $\mathcal{C}$ if $\chi_P \in \mathcal{C}$
- If $A, B \subseteq \mathbb{N}^n$ are in $\mathcal{C}$, then $A \cup B$, $A \cap B$ and $\mathbb{N}^n \setminus A$ are in C

  So if P, Q are predicates of n variables in $\mathcal{C}$, $P \vee Q$, $P \wedge Q$ and $\neg P$ are in $\mathcal{C}$

  Proof: $\chi_{A \cup B}(x) = sg(\chi_A(x) + \chi_B(x))$, $\quad \chi_{A \cap B} = \chi_A(x) \cdot \chi_B(x)$, $\quad \chi_{\mathbb{N}^n \setminus A}(x) = 1 \dot{-} \chi_A(x)$
- The predicates $x = y$, $x \neq y$, $x \leq y$, $x < y$, $x \geq y$, $x > y$ are prim. rec.

  Proof: Note that $\chi_{\neq}(x, y) = sg(|1 - 3|)$ and $\chi_{\geq}(x, y) = sg(x \dot{-} y)$
- Bounded quantifiers:

  Assume P is a pred. of $n+1$ variables in $\mathcal{C}$; then Q, R of $n+1$ variables defined below are in $\mathcal{C}$:

  $Q(x_1, \ldots x_n, z)$ is true if and only if $\exists_{y \leq z}(P(x_1, \ldots, x_n, y)$ is true)

  $R(x_1, \ldots x_n, z)$ is true if and only if $\forall_{y \leq z}(P(x_1, \ldots, x_n, y)$ is true)

  Proof: $\chi_Q(\mathbf{x}, z) = sg(\sum_{y=0}^{z} \chi_P(\mathbf{x}, y))$, and $\chi_R(\mathbf{x}, z) = \prod_{y=0}^{z} \chi_P(\mathbf{x}, y)$

3.4 More Primitive Recursive Functions

3.4.1 Bounded Minimisation

Let P be a pred. of $n+1$ variables. Define $f : \mathbb{N}^{n+1} \to \mathbb{N}$ by:

$$f(\mathbf{x}, z) = \begin{cases} \text{the least } y \leq z \text{ s.t. } P(\mathbf{x}, y) \text{ is true} \\ z + 1 \text{ if no such } y \text{ exists} \end{cases}$$

Then, $f(\mathbf{x}, z) = \mu\ y \leq z\ P(\mathbf{x}, y)$, called *bounded minimisation.* We have that if $P \in \mathcal{C}$ (a prim. rec. closed class), then f is in $\mathcal{C}$.

Proof. Define $g(\mathbf{x}, t) = \prod_{y=0}^{t} sg(1 \dot{-} \chi_P(\mathbf{x}, y) \text{ is true})$. Note that $g(\mathbf{x}, t) = \begin{cases} 0 \text{ if } \exists_{y \leq t} P(x, y) \text{ is true} \\ 1 \text{ if } \forall_{y \leq t} P(x, y) \text{ is false} \end{cases}$

Let $y \leq z$ be the least s.t. $P(\mathbf{x}, y)$ is true.

Then the values of g look like:

t	0	1	...	$y-1$	y	$y+1$	...	z
$g(\mathbf{x}, t)$	1	1	...	1	0	0	...	0

Let $f(\mathbf{x}, z) = \sum_{t=0}^{z} g(\mathbf{x}, t)$, then we will have f as required for bounded minimsation. If there is no such y, then by the definition of g we would have $f(\mathbf{x}, z) = z + 1$

□

3.4.2 Definition By Cases

Let $f_1, \ldots, f_k : \mathbb{N}^n \to \mathbb{N}$ be in prim. rec. closed $\mathcal{C}$ and let $P_1, \ldots, P_k$ be predicates in $\mathcal{C}$ of n variables. Suppose that for each $\mathbf{x} \in \mathbb{N}^n$ *exactly* one of $P_1(\mathbf{x}), \ldots, P_k(\mathbf{x})$ is true. Define $f : \mathbb{N}^n \to \mathbb{N}$ by:

$$f(\mathbf{x}) = f_i(\mathbf{x}) \text{ if } P_i(\mathbf{x}) \text{ is true}$$

Then f is in $\mathcal{C}$.

Proof. $f(\mathbf{x}) = f_1(\mathbf{x}) \cdot \chi_{P_1}(\mathbf{x}) + \cdots + f_k(\mathbf{x}) \cdot \chi_{P_k}(\mathbf{x})$

□

3.4.3 Iteration

Let X be a set, with a partial function $f : X \to X$. The *iterate* of f is the partial function $F : X \times \mathbb{N} \to X$ defined by:

$$\begin{aligned} &F(x, 0) = x \\ &F(x, n+1) = f(F(x, n)) \end{aligned}$$

We have a notion of a function $f : \mathbb{N}^n \to \mathbb{N}$ being in a class $\mathcal{C}$. This can be extended to functions $f : \mathbb{N}^n \to \mathbb{N}^k$ by saying that f is in $\mathcal{C}$ if $\pi_{i,k} \circ f$ is in $\mathcal{C}$ for each $1 \leq i \leq k$.

A class $\mathcal{C}$ is closed under iteration if, whenever $f : \mathbb{N}^n \to \mathbb{N}^n$ is in $\mathcal{C}$, then its iterate $F : \mathbb{N}^{n+1} \to \mathbb{N}^n$ is in $\mathcal{C}$.

Let $\mathcal{C}$ be a prim. rec. closed class. Then if $f : \mathbb{N}^n \to \mathbb{N}^n$ is in $\mathcal{C}$, its iterate $F : \mathbb{N}^{n+1} \to \mathbb{N}^n$ is also in $\mathcal{C}$. So any prim. rec. closed class is closed under iteration.

Proof. This shows only the $n = 1$ case.

Define $f' : \mathbb{N}^3 \to \mathbb{N}$ by $f'(x, y, z) = f(z)$, which is in $\mathcal{C}$.

Then the iterate of f is defined by:

$$\begin{aligned} &F(z, 0) = z \\ &F(z, y+1) = f(F(z, y)) = f'(x, y, F(x, y)) \end{aligned}$$

This is defined by primitive recursion, so F is in $\mathcal{C}$.

□

3.5 Recursive and Partial Recursive Functions

3.5.1 Minimisation

Let $f : \mathbb{N}^{n+1} \to \mathbb{N}$ be a partial function. The function obtained from f by *minimisation* is the partial function $g : \mathbb{N}^n \to \mathbb{N}$ defined by

$$g(\mathbf{x}) = \begin{cases} r & \text{if } f(\mathbf{x}, r) = 0 \text{ and for } s < r, f(\mathbf{x}, s) \text{ is defined and not } 0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

We write $g(\mathbf{x}) = \mu y(f(\mathbf{x}, y) = 0)$. It is also called the μ-operator or unbounded search operator. The function g may be partial, even if f is total, and vice versa.

Note that it is not *quite* accurate to say $g(x) = \mu y(f(\mathbf{x}, y) = 0)$ is the least y s.t. $f(x, y) = 0$; if there is some least y s.t. $f(x, y) = 0$, but $f(x, s)$ is undefined for some $s < y$, then $g(x)$ is undefined.

3.5.2 The Class of Recursive Functions

- A total function $f(\mathbf{x}, y)$ is *regular* if for any $\mathbf{x} \in \mathbb{N}^n$, there exists $y \in \mathbb{N}$ such that $f(\mathbf{x}, y) = 0$
- The regular functions are exactly those to which we can apply minimisation and end up with a total function
- The function g is obtained from f by *regular minimisation* if $g(\mathbf{x}) = \mu y(f(\mathbf{x}, y) = 0)$ where f is regular
- The *class of recursive functions* is the smallest class $\mathcal{C}$ of total functions which is primitively recursively closed and is closed under regular minimisation
- Note that there are recursive functions which are *not* primitive recursive
- Example: the two-argument Ackermann function defined by:

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

- The *class of partial recursive functions* is the smallest class of partial functions which contains the initial functions, and is closed under composition, primitive recursion and minimisation
  - Note that this is *not* a primitively recursively closed class – that term only applies to a class of total functions

4 Equivalence of Partial Recursive and TM Computable Functions

A key theorem is that all partial recursive functions are Turing Machine computable, and vice versa.

4.1 TM Computable Functions Are Partial Recursive

Recall that a partial function $f : \mathbb{N}^n \to \mathbb{N}$ is TM computable if $f = \varphi_{T,n}$ for some numerical TM T.

Let $T = (Q, F, A, I, \tau, q_0)$ be a numerical Turing machine (i.e. deterministic, $F = I = \emptyset$, $A = \{0, 1\}$).

Recall that $\varphi_{T,n}(\mathbf{x}) = \begin{cases} y & \text{if the computation starting with } (q_0, \underline{0}1^{x_1} \ldots 01^{x_n}) \text{ halts with } (q, \underline{0}1^y) \\ \textit{undefined} & \text{otherwise} \end{cases}$

It is convenient to modify T slightly. Add two new states p and h, and the transitions:

- (q, a, p, a, L) for all $(q, a) \in Q \times A$ s.t. no element in τ starts with (q, a)
- (p, a, h, a, R) for all $a \in A$ (i.e. for $a = 0$ and $a = 1$)
- (h, a, p, a, L) for all $a \in A$

Call the new machine T', so $Q' = Q \cup \{p, h\}$, with C' being the set of configurations.

Then T' is still deterministic, and transitions have the form:

$$(q, a, N(q, a), R(q, a), D(q, a)) \in Q' \times A \times Q' \times A \times \{L, R\}$$

where N, R, D are functions on $Q' \times A$.

Then, we number the states such that $Q = \{0, 1, \ldots, r - 1\}$, where $h = 0$ and $p = 1$. We encode $L = 0$ and $R = 1$.

Now, $Q' \times A$ is a finite subset of $\mathbb{N}^2$; put $N(x, y) = R(x, y) = D(x, y) = 0$ for $(x, y) \in \mathbb{N}^2 \setminus (Q' \times A)$. Then, N, R, D are primitive recursive functions $\mathbb{N}^2 \to \mathbb{N}$.

Define $Code : C' \to \mathbb{N}$ by $Code(q, a, \alpha, \beta) = 2^q 3^a 5^{\sigma(\alpha)} 7^{\sigma(\beta)}$, where σ encodes a function in the binary representation of an integer:

$$\sigma(f) = f(0) + 2 \cdot f(1) + 2^2 \cdot f(2) + \ldots$$

Then $Code$ is an injective (one-to-one) function.

There is a primitive recursive function $Next : \mathbb{N} \to \mathbb{N}$ s.t. $Next(Code(c)) = Code(\delta(c))$, for $c \in C'$ where δ is the transition function of T'.

Proof. Let $c = (q, a, \alpha, \beta)$; let $x \in \mathbb{N} = Code(c) = 2^q 3^a 5^{\sigma(\alpha)} 7^{\sigma(\beta)}$.

Then, we express $Next(x) = Code(\delta(c))$ in terms of x.

First, note that $q = \log_2 x$ and $a = \log_3 x$ (here log simply retrieves the exponents, it is not the normal logarithm function from calculus/analysis).

We have that $N(q, a) = N(\log_2 x, \log_3 x)$; the log and N functions are primitive recursive.

There are then two cases, moving left or right:

4.1.1 Move left - $D(q,a)=0$

We have that $\delta(c)=(q',a',\alpha',\beta')$, where $q'=N(q,a)$ and $a'=\beta(0)$.

$Next(x)=Code(\delta(c))=2^{N(q,a)}3^{\beta(0)}5^{\sigma(\alpha')}7^{\sigma(\beta')}$

$\beta(0)=rem(2,\log_7(x))$ where rem is the remainder function (which is prim. rec.).

$\sigma(\alpha')=R(q,a)+2\alpha(0)+2^2\alpha(1)+\cdots=R(\log_2 x,\log_3 x)+2\log_5 x$, where R is prim. rec.

$\sigma(\beta')=\beta(1)+2\beta(2)+2^2\beta(3)+\cdots=quo(2,\sigma(\beta))=quo(2,\log_7 x)$, where quo is the quotient / 'integer division' function (which is prim. rec.).

4.1.2 Move right - $D(q,a)=1$

In this case we have that $\delta(c)=(q',a',\alpha',\beta')$, where $q'=N(q,a)$ and $a'=\alpha(0)$.

$\alpha(0)=rem(2,\log_5 x)$

$\sigma(\alpha')=quo(2,\log_5 x)$

$\sigma(\beta')=R(\log_2 x,\log_3 x)+2\log_7 x$

4.1.3 Conclusion

We can combine both cases using $E(x)=D(\log_2 x,\log_3 x)$. This gives us the functions:

$F_1(x)=N(\log_2 x,\log_3 x)$

$F_2(x)=(1\dot{-}E(x))\cdot rem(2,\log_7 x)+E(x)rem(2,\log_5 x)$

$F_3(x)=(1\dot{-}E(x))\cdot(R(\log_2 x\log_3 x)+2\log_5 x)+E(x)\cdot quo(2,\log_5 x)$

$F_4(x)=(1\dot{-}E(x))\cdot quo(2,\log_7 x)+E(x)\cdot(R(\log_2 x,\log_3 x)+2\log_7 x)$

Clearly each of these is a composition of primitive recursive functions, and so each is primitive recursive.

Then, $Next(x)=2^{F_1(x)}3^{F_2(x)}5^{F_3(x)}7^{F_4(x)}$. This is a composition of exponentiation and functions known to be primitive recursive, so $Next(x)$ is also primitive recursive.

$\square$

Recall that if $f:\mathbb{N}\to\mathbb{N}$ is primitive recursive, then its iterate $F:\mathbb{N}^2\to\mathbb{N}$ is also prim. rec.

Let $\bar{\delta}$ be the iterate of δ. If $Comp$ is the iterate of $Next$, then $Comp(Code(c),t)=Code(\bar{\delta}(c,t))$ for any $c\in C'$ and $t\in\mathbb{N}$.

Proof. Use induction on t.

First, $Comp(Code(c),0)=Code(c)=Code(\bar{\delta}(c,0))$.

Now, assume that $Comp(Code(c),t)=Code(\bar{\delta}(c,t))$ holds.

Then, we have:

$$\begin{aligned}
Comp(Code(c),t+1)&=Next(Comp(Code(c),t))\\
&=Next(Code(\bar{\delta}(c,t)))\\
&=Code(\delta(\bar{\delta}(c,t)))\\
&=Code(\bar{\delta}(c,t+1))
\end{aligned}$$

$\square$

Define the function $In_{T,n} : \mathbb{N}^n \to C'$, such that $In_{T,n}(\mathbf{x})$ returns the initial configuration of T when started with the tape described by $Tape(\mathbf{x})$.

Main theorem: the function $\varphi_{T,n}$ is partial recursive.

Proof. Note $\varphi_{T,n}(\mathbf{x}) = \begin{cases} y & \text{if } \exists t \in \mathbb{N} \text{ s.t. } \bar{\delta}(In_{T,n}(\mathbf{x}), t) = (h, \underline{0}1^y) \text{ for some } y \in \mathbb{N} \\ \textit{undefined} & \text{otherwise} \end{cases}$

Also note that $Code(h, \underline{0}1^y) = 2^0 3^0 5^{1+2+2^2+\cdots+2^{y-1}} 7^0 = 5^{2^y-1}$.

If $\bar{\delta}(In_{T,n}(\mathbf{x}), t) = (h, \underline{0}1^y)$ for some $t, y \in \mathbb{N}$, then we have that:

$Comp(Code(In_{T,n}(\mathbf{x})), t) = Code(\bar{\delta}(In_{T,n}(\mathbf{x}), t)) = 5^{2^y-1}$

Define $\psi : \mathbb{N}^{n+1} \to \mathbb{N}$ by $\psi(\mathbf{x}, t) = Comp(Code(In_{T,n}(\mathbf{x})), t)$.

The composition $Code(In_{T,n}(\mathbf{x}))$ is primitive recursive (from assignments), and $Comp$ is primitive recursive since it is the iterate of the primitive recursive $Next$. Therefore ψ is primitive recursive. Then:

$$\varphi_{T,n}(\mathbf{x}) = \begin{cases} \log_2(1 + \log_5(\psi(\mathbf{x}, t))) & \text{for any } t \in \mathbb{N} \text{ s.t. } \psi(\mathbf{x}, t) = 5^{2^y-1} \text{ for some } y \\ \textit{undefined} & \text{otherwise} \end{cases}$$

e P defined by $P(\mathbf{x}, t)$ is true $\leftrightarrow \psi(\mathbf{x}, t) = S^{s^y-1}$ for some y is primitive recursive. The functions functions F and G defined as follows are then also primitive recursive:

$$F(\mathbf{x}, t) = \log_2(1 + \log_5(\psi(\mathbf{x}, t)))$$
$$G(\mathbf{x}, t) = 1 - \chi_P(\mathbf{x}, t)$$

Then we have that:

$$\varphi_{T,n}(\mathbf{x}) = \begin{cases} F(\mathbf{x}, t) & \text{for any } t \in \mathbb{N} \text{ s.t. } G(\mathbf{x}, t) = 0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

Or equivalently:

$$\varphi_{T,n}(\mathbf{x}) = F(\mathbf{x}, \mu t(G(\mathbf{x}, t) = 0))$$

which is a composition of primitive recursive functions and unbounded minimisation. Therefore $\varphi_{T,n}$ is partial recursive.

□

As it turns out, for a partial function $f : \mathbb{N}^n \to \mathbb{N}$, the following are equivalent:

1. f is partial recursive
2. f is abacus computable
3. f is computable by a register program
4. f is Turing Machine computable

4.2 Other Results in Computability Theory

Another result is that the class of recursive functions is the same as the class of partial recursive functions that are total (not actually a trivial statement!)

4.2.1 The Halting Problem

Let $\mathcal{TM}$ be the set of numerical Turing machines whose set of states is 0, 1, ..., r for some $r \in \mathbb{N}$. Then $\mathcal{TM}$ is countable, and as a corollary, there are countably many partial recursive functions.

An *indexing* of a countable set is an infinite sequence $\psi_0, \psi_1, \ldots$ of elements of S that includes all elements of S (though there may be repetitions).

Important theorem: Let $T_0, T_1, \ldots$ be any indexing of $\mathcal{TM}$. Let $\psi_m := \varphi_{T_m,1}$. Then the function $f : \mathbb{N}^2 \to \mathbb{N}$ defined by:

$$f(x, y) = \begin{cases} 1 & \text{if } \psi_x(y) \text{ is defined} \\ 2 & \text{if } \psi_x(y) \text{ is undefined} \end{cases}$$

is not recursive.

Proof. Assume that f is recursive, and let $g : \mathbb{N} \to \mathbb{N}$ be defined by $g(x) = f(x, x)$ for $x \in \mathbb{N}$. Clearly g is recursive.

Define $\theta : \mathbb{N} \to \mathbb{N}$ by:

$$\theta(x) = \begin{cases} 0 & \text{if } g(x) = 0 \\ \textit{undefined} & \text{if } g(x) = 1 \end{cases}$$

Note that $\theta(x) = \mu y((y + 1) \cdot g(x) = 0)$, so θ is partial recursive.

Now we claim that θ *cannot* be partial recursive.

Note that:

$$\theta(x) = \begin{cases} 0 & \text{if } \psi_x(x) \text{ is undefined} \\ \textit{undefined} & \text{if } \psi_x(x) \text{ is defined} \end{cases}$$

Since θ is partial recursive, $\theta = \psi_i$ for some $i \in \mathbb{N}$, and so $\theta(i) = \psi_i(i)$. Then let the predicate $P(x)$ represent the statement "$\theta(x)$ is defined", or equivalently, "$\psi_x(x)$ is undefined".

Then consider $P(i)$, which is:

$$\theta(i) \text{ is defined} \leftrightarrow \psi_i(i) \text{ is defined} \leftrightarrow \psi_i(i) \text{ is undefined}$$

This is a contradiction. Therefore, the initial assumption (that f is recursive) is false.

□

5 First Order Logic

5.1 First Order Languages: Syntax

- A *first-order language* (FOL) consists of the following symbols:
  - *Logical symbols* $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, =, \forall, \exists$ (common to all FOLs)
  - An infinite set of variables, $x, y, z, \dots$ (also common to all FOLs)
  - Punctuation symbols: parentheses (and) and the comma ',' (also common to all FOLs)
  - A (possibly empty) set of constant symbols (e.g. $0, 1$)
  - A (possibly empty) set of function symbols (e.g. $+, \times, -$)
  - A (possibly empty) set of predicate symbols (e.g. $<$)
- Each function and predicate symbol has an associated arity n
- Only the *non-logical symbols* are specific to the particular language
- A FOL may be specified by giving only the constant, relation and function symbols
  - E.g. the first-order language of arithmetic $\mathcal{L}_A$ consists of the following:
    - The constant symbol 0
    - Unary function symbol S (the successor function)
    - Two binary function symbols $+$ and $\cdot$
- Given an FOL $\mathcal{L}$, an *expression* of $\mathcal{L}$ is a finite sequence of symbols; not all expressions are *formulae*
- A *term* of an FOL is defined inductively:
  - Every constant symbol in $\mathcal{L}$ is a term
  - Every variable symbol in $\mathcal{L}$ is a term
  - If $t_1, \dots, t_n$ are terms and f is an n-ary function symbol in $\mathcal{L}$, then $f(t_1, \dots t_n)$ is a term in $\mathcal{L}$
- An *atomic formula* of an FOL is defined as follows:
  - If t_1 and t_2 are terms, then $t_1 = t_2$ is an atomic formula
  - If F is an n-ary predicate and $t_1, \dots, t_n$ are terms, then $F(t_1, \dots, t_n)$ is an atomic formula
- A *formula* of an FOL is defined inductively:
  - An atomic formula is a formula
  - If ϕ and ψ are both formulae, then so are $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, and $\phi \leftrightarrow \psi$
  - If ϕ is a formula and x is a variable symbol, then $\exists x \phi$ and $\forall x \phi$ are formulae
  - Parentheses should be used as necessary to ensure there is exactly one way of reading a formula
- A variable is *bound* by a quantifier $\forall x$ or $\exists x$ in a formula ϕ if:
  - x is in the scope of the quantifier; and
  - the scope of the quantifier contains no other quantifiers over x with x in their scope
- Any variable which is not bound in a formula ϕ is *free* in ϕ
- A *sentence* of an FOL is a formula with no free variables
- Importantly, an FOL gives no *meaning* to formulae – they are not 'true' or 'false'

5.2 Models: Semantics

- For an FOL $\mathcal{L}$, an $\mathcal{L}$*-structure* or *model* $\mathcal{M}$ consists of the following:
  - A domain or universe: a non-empty set $|\mathcal{M}|$
  - Interpretation for constant symbols: for each constant symbol c of $\mathcal{L}$, an element $c^{\mathcal{M}} \in |\mathcal{M}|$
  - Interpretation for predicate symbols: for each n-ary predicate symbol R of $\mathcal{L}$, an n-ary predicate $R^{\mathcal{M}} \subseteq |\mathcal{M}|^n$
  - Interpretation for function symbols: for each n-ary function symbol f of $\mathcal{L}$, an n-ary function $f^{\mathcal{M}} : |\mathcal{M}|^n \to |\mathcal{M}|$
- A sentence of $\mathcal{L}$ acquires *meaning* when an $\mathcal{L}$-structure $\mathcal{M}$ is given and the sentence is interpreted within $\mathcal{M}$
- We can determine the truth value of a formula ϕ (possibly with free variables) in $\mathcal{L}$-structure $\mathcal{M}$ if a *variable assignment* $\alpha :$ set of variable symbols $\to |\mathcal{M}|$ is given
- For given α, replace all free variables x_i in ϕ by $\alpha(x_i)$, so ϕ becomes a statement in $\mathcal{M}$ which must either be true or false
- We say a formula ϕ is *true in* $\mathcal{M}$ if ϕ is true for **any** variable assignment α
- For a sentence ϕ in $\mathcal{L}$, its truth values does not depend on variable assignment (since there is no free variable). Thus ϕ must be either true or false in $\mathcal{M}$, independent of variable assignment

5.3 Axiomatic Systems & Proof

- A formal axiomatic system comprises:
  - A first-order language
  - Syntactic rules for constructing formulae from the symbols
  - A collection of axioms
  - Rules of inference
- From the axioms we obtain other formulae using the rules of inference, called *theorems*
- A *proof* of a theorem is the process of applying the rules
- A set of *Logical axioms* are common to first-order axiomatic systems
- We may also state theory-specific *non-logical axioms*
- Two logical inference rules are also provided:
  - Modus ponens: $\dfrac{\phi \to \psi, \phi}{\psi}$
  - Generalisation: $\dfrac{\phi}{\forall x \phi}$
- Given T, a 'theory' or (possibly empty) set of non-logical axioms in $\mathcal{L}$, a formula ψ is *provable* in T, denoted $T \vdash \psi$ if there is a finite sequence $\phi_1, \dots, \phi_n$ of formulae such that ϕ_n is equal to ψ and for all i with $1 \leq i \leq n$ we have:
  - ϕ_i is a logical axiom; or
  - $\phi_i \in T$; or
  - There are $j, k < i$ such that ϕ_j is equal to the formula $\phi_k \to \phi_i$; or
  - There is a $j < i$ such that ϕ_i is equal to the formula $\forall x \phi_j$
- If a formula ψ is not provable in T, then we write $T \nvdash \psi$
- A formula ϕ is a *tautology* if $\vdash \phi$ (i.e. it may be proved with no theory-specific axioms)
- We say two formulae ϕ and ψ are *equivalent*, denoted $\phi \equiv \psi$ if $\vdash \phi \leftrightarrow \psi$; that is, if $\phi \leftrightarrow \psi$ is a tautology
- We say a theory T is *consistent* if there is no formula ϕ in $\mathcal{L}$ such that $T \vdash (\phi \wedge \neg\phi)$
  - If T is inconsistent, then for all formulae ψ in $\mathcal{L}$ we have $T \vdash \psi$

    "From contradiction, everything follows"
- We say a theory T is *complete* if for all formulae ϕ in $\mathcal{L}$, $T \vdash \phi$ or $T \vdash \neg\phi$