Preliminaries on Computability and Algorithmic information theory

Lancelot Da Costa & GPT-5.2

Abstract

A concise introduction to computability and algorithmic information theory for mathematicians.

1 Preliminaries

Definition 1.1 (Alphabet)**.** An alphabet Σ is a finite non-empty set.

Definition 1.2 (Words over an alphabet)**.** If Σ is a finite set (alphabet), define Σ^n to be the set of length-n words over Σ and

$$\Sigma^* := \bigcup_{n=0}^{\infty} \Sigma^n$$

to be the set of all finite words over Σ. We write ϵ for the unique word of length 0 (the empty word), so $\Sigma^0 = \{\epsilon\}$. For $x \in \Sigma^*$ we write $|x|$ for its length, i.e. $x \in \Sigma^n \iff |x| = n$. In particular, $\{0,1\}^*$ denotes the set of finite binary strings, and $|p|$ is the bit-length of $p \in \{0,1\}^*$.

Definition 1.3 (Languages)**.** A *language* over an alphabet Σ is a subset $L \subseteq \Sigma^*$.

Definition 1.4 (Prefix and strict prefix)**.** Let Σ be an alphabet and let $p, q \in \Sigma^*$. We say that p is a *prefix* of q if there exists $r \in \Sigma^*$ such that $q = pr$. If additionally $r \neq \epsilon$ (equivalently, $p \neq q$), then p is a *strict prefix* of q.

Definition 1.5 (Prefix-free languages)**.** A language $L \subseteq \Sigma^*$ is *prefix-free* (a.k.a. *self-delimiting*) if there do not exist distinct $p, q \in L$ such that p is a prefix of q. *Example:* Over $\Sigma = \{0,1\}$, the language $L = \{0, 01\}$ is not prefix-free because 0 is a strict prefix of 01. In contrast, the set $T = \{0, 10, 110\}$ is prefix-free.

Definition 1.6 (Partial functions)**.** Given sets A, B, a *partial function* $f : A \rightharpoonup B$ is a function $f : \mathrm{dom}(f) \to B$ with domain $\mathrm{dom}(f) \subseteq A$. For $a \notin \mathrm{dom}(f)$ we say $f(a)$ is *undefined*.

Definition 1.7 (Prefix-free functions)**.** A partial function $f : \Sigma^* \rightharpoonup \Sigma^*$ is *prefix-free* (a.k.a. *self-delimiting*) if its domain $\mathrm{dom}(f) := \{x \in \Sigma^* : f(x) \text{ is defined}\}$ is a prefix-free language (in the sense of Definition 1.5).

2 Turing machines: the mechanics of computation

Definition 2.1 (One-tape Turing machine)**.** A deterministic one-tape Turing machine is a tuple $M = (Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states,
- Σ is an alphabet (the *input alphabet*), e.g. $\Sigma = \{0,1\}$,
- Γ is an alphabet (the *tape alphabet*) with $\Sigma \cup \{\sqcup\} \subseteq \Gamma$, where $\sqcup \in \Gamma$ is the *distinguished blank symbol*,
- $q_0 \in Q$ is a start state,
- $F \subseteq Q$ is a set of halting states, and
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function, where L (resp. R) means "move the head one cell left" (resp. right).

A *tape* over Γ is a function $t : \mathbb{Z} \to \Gamma$ such that $\mathrm{supp}_{\sqcup}(t) := \{i \in \mathbb{Z} : t(i) \neq \sqcup\}$ is a finite set ("all but finitely many tape cells are blank"). Let $\mathcal{T}_\Gamma$ denote the set of all such tapes. A *configuration* is a triple $(q, h, t) \in Q \times \mathbb{Z} \times \mathcal{T}_\Gamma$ where q is the state, t is the tape contents and h is the head position. If $q \in Q \setminus F$ and $\delta(q, t(h)) = (q', a', d)$, define the next configuration to be (q', h', t') where $t'(h) = a'$, $t'(i) = t(i)$ for $i \neq h$, and $h' = h - 1$ if $d = L$ while $h' = h + 1$ if $d = R$. So the transition function alters the tape at the previous location and then moves left or right. In particular the condition $t \in \mathcal{T}_\Gamma$ is preserved by the transition rule.

Given an input $x = x_0 x_1 \cdots x_{n-1} \in \Sigma^*$, define the initial configuration to be $(q_0, 0, t_x)$ where $t_x(i) = x_i$ for $i \in \{0, 1, \ldots, n-1\}$ and $t_x(i) = \sqcup$ otherwise. Iterating the transition rule induces a discrete-time evolution of configurations. We write $M(x) \downarrow$ if this evolution reaches some configuration with state in F (halts) and $M(x) \uparrow$ otherwise (diverges).

Definition 2.2 (Acceptance, rejection, and decision)**.** Fix a Turing machine $M = (Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F)$. An *accepting set* is a subset $F_{\text{acc}} \subseteq F$; the *rejecting set* is then $F_{\text{rej}} := F \setminus F_{\text{acc}}$. We say M *accepts* an input x if $M(x) \downarrow$ and the halting state lies in F_{acc}, and M *rejects* x if $M(x) \downarrow$ and the halting state lies in F_{rej}. For a language $L \subseteq \Sigma^*$, we say M *decides* L if $M(x) \downarrow$ for all $x \in \Sigma^*$ and M accepts x iff $x \in L$ (equivalently, M rejects x iff $x \notin L$). *Intuition:* deciding L means membership in L can be computed by an algorithm that always halts and returns a yes/no answer.

Remark 2.1 (Why allow both left and right moves?)*.* Allowing the head to move both left and right is part of what makes the Turing machine model *universal*. If one restricted to machines that only ever move right, the model never revisits earlier tape cells which prevents it from expressing all computable functions, e.g. it can't even decide whether a binary string is a palindrome.

Example 2.1 (A simple one-tape Turing machine)**.** Let $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$. Define a Turing machine that computes the bitwise complement $x \mapsto \bar{x}$ (flip $0 \leftrightarrow 1$) and halts: let $Q = \{q_0, q_h\}$, $F = \{q_h\}$, and define δ on $Q \setminus F = \{q_0\}$ by

$$\delta(q_0, 0) = (q_0, 1, R), \qquad \delta(q_0, 1) = (q_0, 0, R), \qquad \delta(q_0, \sqcup) = (q_h, \sqcup, R).$$

Starting at the leftmost input symbol, this machine scans right, flips each bit, and halts upon reaching the first blank.

Example 2.2 (A palindrome-decider (informal))**.** Let $\text{PAL} := \{x \in \{0, 1\}^* : x \text{ is a palindrome}\}$. There is a one-tape Turing machine that decides PAL using the following routine. Extend the tape alphabet to $\Gamma = \{0, 1, \sqcup, X_0, X_1\}$, where X_0 (resp. X_1) marks a cell that originally contained 0 (resp. 1) and has already been matched. Repeatedly: scan right from the left end to find the leftmost unmarked symbol $b \in \{0, 1\}$ and replace it by X_b; then scan to the right end, move left to find the rightmost unmarked symbol (if none exists, accept), and compare it to b. If it matches, replace it by X_b and return to the left; if it mismatches, reject. This halts and accepts exactly the palindromes because each iteration matches the current leftmost and rightmost unmatched symbols. *Informal example*: $01010 \to X_0 1010 \to X_0 101 X_0 \to X_0 X_1 01 X_0 \to X_0 X_1 0 X_1 X_0 \to X_0 X_1 X_0 X_1 X_0 \to$ accept.

3 Computability: function computation

Definition 3.1 (Output convention)**.** Fix a one-tape Turing machine $M = (Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F)$. Write $\mathcal{T}_\Gamma$ for the set of tapes defined in Definition 2.1, and define the set of *halting configurations* $\text{Halt}(M) := F \times \mathbb{Z} \times \mathcal{T}_\Gamma$. An *output convention* for M is a function

$$\text{out} : \text{Halt}(M) \to \Sigma^*.$$

Example: two common choices are:

- (Head-readout.) If the halting tape is known (by construction of M) to contain only Σ-symbols on a contiguous block starting at the head, define $\text{out}(q, h, t)$ to be that block.
- (Origin-readout.) If the halting tape is known to contain only Σ-symbols on cells $0, 1, \ldots, r$ and blanks elsewhere, define $\text{out}(q, h, t) := t(0)t(1)\cdots t(r)$ where $r := \max\{i \in \mathbb{Z} : t(i) \neq \sqcup\}$ (with $r = -1$ interpreted as output ϵ).

Definition 3.2 (Computable and partial computable functions)**.** Given a Turing machine M with input alphabet Σ and output convention out, define the induced *partial output function* $\text{out}_M : \Sigma^* \rightharpoonup \Sigma^*$ as follows: for $x \in \Sigma^*$, if $M(x) \downarrow$ and the resulting halting configuration is $(q, h, t) \in \text{Halt}(M)$, set

$$\text{out}_M(x) := \text{out}(q, h, t),$$

and if $M(x) \uparrow$ then leave $\text{out}_M(x)$ undefined.

A (total) function $f : \Sigma^* \to \Sigma^*$ is *computable* if there exists a Turing machine M with input alphabet Σ such that out_M is total and $\text{out}_M(x) = f(x)$ for all $x \in \Sigma^*$. A partial function $f : \Sigma^* \rightharpoonup \Sigma^*$ is *partial computable* if there exists a Turing machine M with input alphabet Σ such that $\text{out}_M = f$ as partial functions, i.e. $\text{dom}(\text{out}_M) = \text{dom}(f)$ and whenever $x \in \text{dom}(f)$ one has $\text{out}_M(x) = f(x)$.

Theorem 3.1 (Robustness under computable post-processing)**.** *If $f : \Sigma^* \rightharpoonup \Sigma^*$ is partial computable and $g : \Sigma^* \to \Sigma^*$ is total computable, then $g \circ f : \Sigma^* \rightharpoonup \Sigma^*$ is partial computable.*

This is a standard closure property of partial computable functions; see e.g. Cutland, *Computability: An Introduction to Recursive Function Theory* (1980), or Rogers, *Theory of Recursive Functions and Effective Computability* (1967/1987).

4 Kolmogorov complexity

In this section we specialise to the *binary* alphabet $\{0, 1\}^*$.[1] We work with two standard variants of Kolmogorov complexity:

[1] This is standard and entails no loss of generality: any alphabet Σ can be encoded into $\{0, 1\}^*$ by a fixed computable injective map, changing Kolmogorov complexities by at most an additive $O(1)$ term (see **??**).

- *plain* Kolmogorov complexity (denoted C), and
- *prefix-free* (a.k.a. *self-delimiting*) Kolmogorov complexity (denoted K).

Remark 4.1. Some ML papers use the symbol K to denote the *plain* notion (what we call C); here we follow the standard Algorithmic Information Theory convention C for plain and K for prefix-free.

Definition 4.1. Given a partial computable function $f : \{0,1\}^* \rightharpoonup \{0,1\}^*$, the *plain Kolmogorov complexity relative to* f is the function

$$C_f : \{0,1\}^* \to \mathbb{N} \cup \{\infty\}, \qquad C_f(x) \;:=\; \min\{\, |p| : f(p) = x \,\}.$$

Example: $C_f(x) = \infty$ if x is not in the range of f.

Definition 4.2. Given a prefix-free partial computable function f (in the sense of Definitions 1.7 and 3.2), its *prefix-free* (a.k.a. *self-delimiting*) Kolmogorov complexity is

$$K_f : \{0,1\}^* \to \mathbb{N} \cup \{\infty\}, \qquad K_f(x) \;:=\; \min\{\, |p| : f(p) = x \,\}.$$