# ALGORITHMIC PROBABILITY IN HYPOTHESIS SELECTION

CONOR L. MAHANY

Abstract. In this paper we explore the relation between computability theory and hypothesis identification via the notion of Kolmogorov complexity. Necessary concepts such as Turing machines, partial recursive functions, and prefix codes are defined in Sections 1-4 before we construct a universal semimeasure which intuitively is "maximally ignorant". This construction and its relation to the prefix complexity allow us to formulate in Section 7 the ideal MDL (Minimum Description Length) principle of hypothesis identification, which requires neither a prior over hypotheses nor conditional priors over the sample space for each hypothesis.

Contents



## 1. Introduction

The principal problem which this paper addresses is, given some data $D$, to identify a model or hypothesis $H$ which best "explains" the data. In this context the "data" $D$ will be a binary string and the "hypothesis" $H$ will often be some

*Date*: 16 September 2013.

collection of binary strings which contains $D$ or a parameter value which underlies the generation of the data.

**Example 1.1.** Suppose that one is flipping a coin, and one knows that it is either a "loaded" coin, in the sense that it always comes up heads, or a "fair" coin, in the sense that heads and tails each occur with probability $\frac{1}{2}$. These are our two hypotheses. Now suppose that one flips this coin $n$ times, and observes $n$ heads consecutively. If $n$ is relatively large (e.g. greater than 4 or 5) and if the two hypotheses are *a priori* equally plausible then we know, intuitively, that the coin is probably loaded, which is to say that we identify this hypothesis ("the coin is loaded") as the most probable given the generated data.

The "intuition" from which we drew above is none other than Bayesian inference. *Bayes' rule* says that for two events $D \in \mathcal{D}$ and $H \in \mathcal{H}$, a prior probability measure $P$ over the class $\mathcal{H}$ of hypotheses, and for each $H \in \mathcal{H}$ a conditional probability measure $\Pr(-|H)$ over the class $\mathcal{D}$, we have the inverse conditional probability $P(H|D)$ — or likelihood of the hypothesis $H$ given the data $D$ — is defined by

$$P(H|D) = \frac{P(D|H)\,P(H)}{P(D)}. \tag{1.2}$$

The principle of Bayesian inference selects the hypothesis $H_B \in \mathcal{H}$ for which Equation 1.2 is maximized, or in other words the most likely hypothesis given the observed data. Bayesian inference aptly characterizes the "intuitive" at the cost of reliance upon a prior distribution over the class of hypotheses under consideration. The conclusion of the example (that the coin is loaded) follows for large $n$ only when the prior probabilities $P(H_1)$ and $P(H_2)$ for the two hypotheses are roughly equal. In this way the conclusion depends upon the prior distribution, which, in situations of statistical inference, is not always available or to be trusted.

Ideal Minimum Description Length — henceforth ideal MDL — by contrast, is an approach to hypothesis identification which does not require a prior distribution.

To define ideal MDL and relate it to Bayesian inference we will begin in Section 2 by reviewing some conventions regarding binary strings. We will then introduce the necessary computability theory in Section 3 by defining Turing machines and partial recursive functions and constructing a universal Turing machine.

We will then proceed to consider prefix codes and a special sort of Turing machine called a prefix machine. Kraft's inequality in Section 4.1 describes the relation between prefix codes and semimeasures. After explicitly constructing a universal prefix machine we will define the prefix complexity.

We then (in Section 5.3) will construct a lower semicomputable discrete semimeasure $m(x)$ which is universal in the sense that for any other lower semicomputable discrete semimeasure $P(x)$ there is a constant $c_P > 0$ such that $m(x) \geq c_P \cdot P(x)$ for all $x$. When $m(x)$ is used as a prior this universality condition can be interpreted to mean that $m(x)$ is "maximally ignorant". The somewhat technical Theorem 5.23 in Section 5.4 describes a remarkable relation between the function $m(x)$ and the prefix complexity.

In Section 6 we formalize the notion of randomness of finite strings by introducing $P$-tests and universal $P$-tests. We proceed to show that we may define a universal $P$-test in terms of the universal lower semicomputable discrete semimeasure.

Finally in Section 7 we introduce ideal MDL, which is defined in terms of prefix complexity. Using Theorem 5.23 we show that for certain classes $\mathcal{H}$ and for large

enough data $D \in \mathcal{D}$ the hypothesis $H_M \in \mathcal{H}$ selected by ideal MDL and the hypothesis $H_B \in \mathcal{H}$ selected by Bayesian inference coincide.

## 2. Binary strings

The purpose of this section is primarily to introduce notation and establish conventions.

**Definition 2.1.** For each $n \in \mathbb{N}$ let $\{0,1\}^n = \{x_1 \cdots x_n \,|\, x_i \in \{0,1\}, i = 1, ..., n\}$. We define the *set of binary strings* to be

$$\{0,1\}^* = \bigcup_{n \in \mathbb{N} \cup \{0\}} \{0,1\}^n.$$

Let $\epsilon$ denote the empty string, which is the unique string such that $l(\epsilon) = 0$. Letting $\cdot \colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ denote concatenation in $\{0,1\}^*$ the 3-tuple $(\{0,1\}^*, \cdot, \epsilon)$ defines a monoid (i.e. with $\epsilon$ the identity element).

Let the $b \colon \mathbb{N} \cup \{0\} \to \{0,1\}^*$ be the bijection defined (lexicographically) by $b(0) = \epsilon, b(1) = 0, b(2) = 1, b(3) = 00, b(4) = 01$, etc. The function $b$ is related to the standard binary representation $\text{bin} \colon \mathbb{N} \cup \{0\} \to \{0,1\}^*$ (which goes $0, 1, 10, 11, 100$, etc) in that $b(n)$ equals $\text{bin}(n+1)$ without its leading 1.

Let $l \colon \{0,1\}^* \to \mathbb{N} \cup \{0\}$ be the map which assigns to each binary string its *length*, so that $l(\epsilon) = 0$ and $l(x_1 ... x_n) = n$ for all $x = x_1 \cdots x_n \in \{0,1\}^*$. Thus for each $n \in \mathbb{N}$ we have that $\{0,1\}^n$ is the set of binary strings of length $n$. Notice that $l(b(i)) = \lfloor \log_2(i+1) \rfloor$ for each $i \in \mathbb{N} \cup \{0\}$. We denote the cardinality of a set $A$ by $d(A)$. The function log will always mean $\log_2$.

**Definition 2.2.** A *partial function* $f \colon X \to Y$ is a function $\overline{f} \colon \text{dom}(f) \to Y$ where $\text{dom}(f) \subseteq X$.

## 3. Basic computability theory

In this section we define Turing machines and partial recursive functions so that we may formalize the intuition that the complexity of a binary string $D$ ought to be the length of the shortest "description" of $D$ or "program" that computes $D$.

**Definition 3.1.** A *deterministic Turing machine* $T$ is a 7-tuple

$$T = (Q, \Gamma, B, \Sigma, \delta, q_0, F)$$

satisfying the conditions

- $B \in \Gamma$,
- $\Sigma \subseteq \Gamma \setminus \{B\}$,
- $\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$,
- $q_0 \in Q$,
- $F \subseteq Q$, and
- $Q, \Gamma$ are both finite.

We call $Q$ the *set of states*, $\Gamma$ the *tape alphabet*, $\Sigma$ the *input alphabet*, $B$ the *blank symbol*, $\delta$ the *transition function*, $q_0$ the *initial state* and $F$ the set of *halting states* of the Turing machine $T$.

*Remark* 3.2. We will restrict our attention only to those Turing machines with tape alphabet $\Gamma = \{0, 1, B\}$ and input alphabet $\Sigma = \{0, 1\}$.

In this paper the term "Turing machine" will always mean deterministic Turing machine. Intuitively, the transition function of a Turing machine determines its behavior on any input in $\{0,1\}^*$. Let $T$ be any Turing machine and let $w \in \{0,1\}^*$. The Turing machine $T$ begins its computation on $w$ with the string $w = w_1 \cdots w_n \in \{0,1\}^*$ written on its tape, with all other squares blank, with the "tape head" pointed to the leftmost square of $w$, and in the initial state $q_0 \in Q$.

We interpret the 3-tuple $\delta(q_i, s_a) = (q_j, s_b, D)$ — with $D \in \{L, R\}$, $q_i, q_j \in Q$, and $s_a, s_b \in \Gamma$ — to mean that if the tape head of $T$ points to the symbol $s_a$ and if $T$ is in state $q_i$, then the Turing machine $T$ will change $s_a$ to $s_b$ on its tape, will enter state $q_j$, and will move left if $D = L$ and right if $D = R$.

Repeatedly applying the transition function $\delta$ of the Turing machine $T$ given the input $w \in \{0,1\}^*$ may eventually reach some state $q \in F$, in which case $T$ is said to *halt* and we define the output $T(w)$ of $T$ on $w$ to be the the longest string $s$ uninterrupted by blanks on the tape of $T$ and connected to the tape cell to which $T$ points once $T$ has halted. Otherwise, $T$ will indefinitely continue computing and never enter a halting state, in which case the output $T(w)$ is not defined.

**Definition 3.3.** For any Turing machine $T$ the *language* $L(T) \subseteq \{0,1\}^*$ of $T$ is the collection of strings $w \in \{0,1\}^*$ for which the Turing machine $T$, given input $w$ in the sense above, eventually enters a halting state $q \in F$.

**Definition 3.4.** For any Turing machine $T$ define the partial function $f_T \colon \{0,1\}^* \to \mathbb{N} \cup \{0\}$ by $f_T(w) = b^{-1}(T(w))$. A *partial recursive function* is a partial function $\phi \colon \{0,1\}^* \to \mathbb{N} \cup \{0\}$ such that $\phi = f_T$ for some Turing machine $T$. A *total recursive function* is a partial recursive function that is defined on all of $\{0,1\}^*$. We call $f_T$ the *partial recursive function associated to* $T$.

Since the bijection $b$ from Section 2 is total recursive we may extend Definition 3.4 to partial functions $\phi \colon \mathbb{N} \to \mathbb{N}$ by requiring instead that $\phi = f_T \circ b$ for some Turing machine $T$. We may do this similarly for partial functions $\mathbb{N} \to \{0,1\}^*$ and $\{0,1\}^* \to \{0,1\}^*$.

**Definition 3.5.** Let $\langle -, - \rangle \colon \mathbb{N} \cup \{0\} \times \mathbb{N} \cup \{0\} \to \{0,1\}^*$ be a total recursive bijection. For any Turing machine $T$ define $f_T^2 \colon \{0,1\}^* \times \{0,1\}^* \to \mathbb{N} \cup \{0\}$ by $f_T(n,l) = b^{-1}(T(\langle n, l \rangle))$.

One can iterate this process (e.g. by $\langle -, \langle -, - \rangle \rangle$) to associate an $n$-ary partial recursive function to each Turing machine $T$ for any $n \in \mathbb{N}$.

**Definition 3.6.** An *enumeration* of a class $\mathcal{C} \subseteq \{0,1\}^*$ is a bijection $f \colon \mathbb{N} \to \mathcal{C}$. An *effective enumeration* is an enumeration that is total recursive.

The finiteness of the states and symbols of the Turing machine allows us to encode each Turing machine with a finite binary string. Suppose that $T$ is a Turing machine with states $Q$ and $|Q| = n$ and tape alphabet $\Gamma = \{0,1\}^* \cup \{B\}$. The transition function

$$\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

uniquely determines $T$ and thus $T$ may be represented by at most $(6n)^{3n}$ 5-tuples. If, for instance, we represent the rule $\delta(q_i, s_a) = (q_j, s_b, D)$ with the binary string $t_{i,a} = 0^i 1 0^a 1 0^j 1 0^b 1 0^{D_k}$, where $D_k = 1$ if $D = L$ and $D_k = 2$ if $D = R$, then we may encode the entire Turing machine $T$ with the binary string

$$w_T = t_{1,1} 1 1 t_{1,2} 1 1 \cdots 1 1 t_{1,m} 1 1 t_{2,1} 1 1 \cdots 1 1 t_{n,m},$$

which is of length at most $nm(2n+2m+6)+2(nm-1)$, for $m=|\Gamma|=3$. In this way each Turing machine is represented by at least one binary string in $\{0,1\}^*$. Since any permutation of the strings $t_{i,a}$ in $w_T$ would yield the same transition function $\delta$ this representation is far from unique.

The bijection $b$ is an effective enumeration of $\{0,1\}^*$. For each Turing machine $T$, there is a binary string $w_T$ which represents its transition function, and hence there is some $j\in\mathbb{N}$ such that $w_T=b(j)$.

Consequently we may construct an effective enumeration $T_1,T_2,...$ of Turing machines by restricting the enumeration $b(1),b(2),...$ of $\{0,1\}^*$ to those strings of the form $t_1 11 t_2 11\cdots 11 t_r$ where $t_i=0^{i_1}10^{i_2}10^{i_3}10^{i_4}10^{i_5}$ for $i_1,i_2,i_3,i_4\in\mathbb{N}$ and $i_5\in\{1,2\}$. For any Turing machine $T$ there is an index $k\in\mathbb{N}$ such that $T=T_k$.

Though we are only considering Turing machines with input alphabet $\{0,1\}$ and tape alphabet $\{0,1,B\}$ this is not for present purposes a restrictive assumption.

**Definition 3.7.** Let $f\colon \mathbb{N}\cup\{0\}\times\{0,1\}^*\to\{0,1\}^*$ be a total recursive bijection. A *universal Turing machine* $U$ is a Turing machine such that for any Turing machine $T$ there exists a number $n_T\in\mathbb{N}$ such that $U(f(n_T,x))=T(x)$ for all $x\in L(T)$.

Note that Definition 3.7 depends upon a choice of bijective pairing function $f\colon \mathbb{N}\cup\{0\}\times\{0,1\}^*\to\{0,1\}^*$. Intuitively a universal Turing machine is a Turing machine which can "simulate" any other Turing machine.

One can construct a relatively simple universal Turing machine $V$ from the enumeration of Turing machines above as follows. For any Turing machine $T$ we already know that there exists a binary string $w_T\in\{0,1\}^*$ which unambiguously represents the transition function of $T$. Suppose that $b(n)=w_T$, that is, that $w_T$ is the $n$-th binary string. Given the input $(n_T,x)$ the machine $U$ initially sets a counter to 0 and then enumerates the strings in $\{0,1\}^*$, checking whether each string is a Turing machine: if it is, the machine increments its counter by 1, and if the counter is less than $n_T$ then it continues enumerating strings in $\{0,1\}^*$. Once the counter reaches $n_T$ the machine $V$ then simulates $T_{n_T}=T$ on $x$ using the representation of its transition function.

**Definition 3.8.** An $n$-ary *universal partial recursive function* $\psi$ is an $(n+1)$-ary partial recursive function $\psi\colon\mathbb{N}^{n+1}\to\mathbb{N}$ such that for any $n$-ary partial recursive function $\phi$ there exists some $i\in\mathbb{N}$ such that

$$\psi(i,m_1,...,m_n)=\phi(m_1,...,m_n)$$

for all $m_1,...,m_n\in\mathbb{N}$.

It follows from these definitions that the $(n+1)$-ary partial function associated to a universal Turing machine is an $n$-ary universal partial recursive function.

*Remark* 3.9. For the remainder of this paper we fix an effective enumeration of Turing machines $T_1,T_2,...$, and the corresponding enumeration of partial functions $\phi_1,\phi_2,...$ (i.e., such that $f_{T_i}=\phi_i$).

We know that effective enumerations of Turing machines exist because the above, explicit enumeration is effective, which follows since the bijection $b$ from Section 2 is an effective enumeration of $\{0,1\}^*$.

**Definition 3.10.** Let $\alpha\colon\mathbb{N}\cup\{0\}\to\mathbb{Q}$ and $\langle -,-\rangle\colon\mathbb{N}\cup\{0\}\times\mathbb{N}\cup\{0\}\to\{0,1\}^*$ be total recursive bijections. Define the partial function $g_T\colon\mathbb{N}\cup\{0\}\times\mathbb{N}\cup\{0\}\to\mathbb{Q}$ by $g_T(n,l)=\alpha\left(b^{-1}(T(\langle n,l\rangle))\right)$, and define the partial function $h_T\colon\mathbb{N}\to\mathbb{R}$ by

$h_T(x) = \lim_{n\to\infty} g_T(x,n)$, when the limit exists. A *real-valued partial recursive function* is a partial function $h\colon \mathbb{N} \to \mathbb{R}$ such that $h = h_T$ for some Turing machine $T$. We call $h_T$ the *real-valued partial recursive function associated to* $T$.

## 4. Basic coding theory and prefix machines

There is an intimate relation between prefix codes and discrete semimeasures which is described by Kraft's inequality. To formulate complexity in a way which may be related to the universal lower semicomputable discrete semimeasure of Section 5.3 we will introduce and subsequently work with a technical variant of the Kolmogorov complexity called the *prefix complexity*.

### 4.1. **Prefix codes.**

**Definition 4.1.** A collection $A \subset \{0,1\}^*$ of binary strings is called *prefix-free* if for all $x, y \in A$ we have that $x = yz$ for some $z \in \{0,1\}^*$ implies that $z = \epsilon$, the empty string.

In other words the collection $A \subset \{0,1\}^*$ is prefix-free if it satisfies the condition that $x$ is not a prefix of $y$ for any distinct $x, y \in A$.

**Definition 4.2.** Let $D\colon \{0,1\}^* \to \mathbb{N}$ be a partial function. We call the domain of $D$ the set of *code words*, and we call the image of $D$ the set of *source words*. If $D(x) = n$ then we say that $x$ is a code word for the source word $n$, and $D$ is the *decoding function*. If the domain of $D$ is prefix-free then $D$ defines a *prefix code*.

**Theorem 4.3.** *(Kraft's inequality) If $l_1, l_2, ...$ are a finite or infinite collection of natural numbers then there is a prefix code with $l_1, l_2, ...$ as the lengths of its code words if and only if*

$$\sum_{n\in\mathbb{N}} 2^{-l_n} \leq 1.$$

*Proof.* ($\Rightarrow$). Note that there is a bijection between finite strings $x \in \{0,1\}^*$ and subsets of $[0,1) \subset \mathbb{R}$ of the form $\Gamma_x = [0.x, 0.x + 2^{-l_x})$. Let $x_1, x_2, ...$ be a prefix code with these lengths, and note that

$$\Gamma_{x_i} \cap \Gamma_{x_j} = \emptyset$$

for all $i \neq j$. Since $\Gamma_{x_i} \subset [0,1)$ for all $i$ we have that

$$\sum_{n\in\mathbb{N}} 2^{-l_n} = \sum_{n\in\mathbb{N}} |\Gamma_{x_n}| = \left| \bigcup_{n\in\mathbb{N}} \Gamma_{x_n} \right| \leq 1.$$

($\Leftarrow$). By the well-ordering principle of $\mathbb{N}$ — which is equivalent to the axiom of choice — we may select a least element of $\{l_1, l_2, ...\}$, say $m_1$, and we may then select a least element of $\{l_1, l_2, ...\} \setminus \{m_1\}$, and we may continue indefinitely in this manner until $m_1, m_2, ...$ is nondecreasing, and this rearrangement does not affect the value of the sum. Let

$$I_n = \left[ \sum_{i=1}^{n-1} 2^{-m_i}, \sum_{i=1}^{n} 2^{-m_i} \right)$$

for each $n$ and observe that since $m_1, m_2, ...$ is nondecreasing each $I_n$ is of the form $\Gamma_{x_n}$ for some $x_n \in \{0,1\}^*$ with $l(x_n) = m_n$. The sequence $x_1, x_2, ...$ defines a prefix code via $x_i \mapsto i$. □

4.2. **Binary intervals.** As suggested by the proof of Kraft's inequality there is a bijection between elements of $x \in \{0,1\}^*$ and certain (half-open) intervals $\Gamma_x \subseteq [0,1)$. Intuitively the binary interval $\Gamma_x$ corresponds to all infinite binary strings $y$ such that $x$ is a prefix of $y$. A simple example is $\Gamma_0 = [0, \frac{1}{2})$, where numbers $0.x_1x_2\cdots$ correspond to strings $x_1x_2\cdots$ with 0 as a prefix, that is, with $x_1 = 0$.

**Definition 4.4.** Let $\{0,1\}^\infty$ denote the collection of infinite binary sequences. A *binary interval* is a subset $I \subset [0,1)$ of the form

$$I = \Gamma_x := \{0.xw : w \in \{0,1\}^\infty\}$$

for some $x = x_1 \cdots x_n \in \{0,1\}^*$. We call $\Gamma_x$ the binary interval associated to $x$.

As the name implies each binary interval $\Gamma_x$ for $x = x_1 \cdots x_n$ may be written as the interval

$$\begin{aligned}&\left[0.x, 0.x + 2^{-l(x)}\right)\\ &= \left[\frac{2^{n-1}x_1 + \cdots + 2^0 x_n}{2^n}, \frac{(2^{n-1}x_1 + \cdots + 2^0 x_n) + 1}{2^n}\right).\end{aligned}$$

**Lemma 4.5.** *If $I \subset [0,1)$ is a nonempty interval and if*

$$\begin{aligned}x_I &= \min\left\{\arg\max_{x\in\{0,1\}^*}\{2^{-l(x)} \colon \Gamma_x \subset I\}\right\}\\ &= \min\left\{\arg\min_{x\in\{0,1\}^*}\{l(x) \colon \Gamma_x \subset I\}\right\}\end{aligned}$$

*is the string which corresponds to the leftmost of the largest binary intervals contained in $I$ then $I$ may be covered by at most four binary intervals of length $2^{-l(x_I)}$.*

*Proof.* Let $x_I - 1 = b(b^{-1}(x_I) - 1)$. Observe first that $\Gamma_{x_I-1} \nsubseteq I$ since otherwise $\Gamma_{x_I-1}$ would be binary interval contained in $I$ that is further left than $\Gamma_{x_I}$, which would contradict the definition of $x_I$.

If the string $x_I$ ends with a 0 and if $\Gamma_{x_I+1} \subset I$ then observe that $\Gamma_s = \Gamma_{x_I} \cup \Gamma_{x_I+1} \subseteq I$ for some binary string $s$ with $l(s) < l(x_I)$, which contradicts the definition of $x_I$. Hence $I$ is covered by three binary intervals of size $2^{-l(x_I)}$, namely $\Gamma_{x_I-1}, \Gamma_{x_I}$, and $\Gamma_{x_I+1}$.

If the string $x_I$ ends instead with a 1 and if $\Gamma_{x_I+1} \cup \Gamma_{x_I+2} \subseteq I$ then similarly $\Gamma_s = \Gamma_{x_I+1} \cup \Gamma_{x_I+2} \subseteq I$ for some binary string $s$ with $l(s) < l(x_I)$, which contradicts the definition of $x_I$. Hence $I$ is covered by four binary intervals of size $2^{-l(x_I)}$, namely $\Gamma_{x_I-1}, \Gamma_{x_I}, \Gamma_{x_I+1}$ and $\Gamma_{x_I+2}$. □

4.3. **Prefix machines.**

**Definition 4.6.** A *partial recursive prefix function* is a partial recursive function $\phi \colon \{0,1\}^* \to \mathbb{N}$ such that the domain of $\phi$ is prefix-free.

**Definition 4.7.** A *prefix machine* is a Turing machine $T$ for which $L(T)$ is prefix-free.

We will modify the standard enumeration of Turing machines — and, correspondingly, of partial recursive functions — so that we have an (effective) enumeration of prefix machines — and therefore, correspondingly, of partial recursive prefix functions.

**Proposition 4.8.** *There exists an effective enumeration of prefix machines.*

*Proof.* We will first define an algorithm which modifies each partial recursive function $\phi\colon \{0,1\}^* \to \mathbb{N}$ so that it is a partial recursive prefix function, and then we will apply this algorithm to an effective enumeration of Turing machines.

By "compute a step of the Turing machine $M$ on $w$" we mean apply the transition function of $M$ in the computation of $M(w)$.

**Data**: a partial recursive function $\phi\colon \{0,1\}^* \to \mathbb{N} \cup \{0\}$ and some $x$ in the domain of $\phi$
**Result**: $\phi'(x) \in \mathbb{N}$
**begin**
  Let $T$ be a Turing machine for which $f_T = \phi$
  Let $p := x$
  Let $l := 1$
  **for** $i = 1, 2, ...$ **do**
    **for** $j = 1, ..., i$ **do**
      Compute a step of $T$ on $pb(j)$
      **if** $T$ has halted on $pb(j)$ **then**
        **if** $b(j) = \epsilon$ **then**
          Define $\phi'(p) = T(p)$
        **else**
          Let $p := px_l$
          Let $l := l + 1$
        **end**
      **end**
    **end**
  **end**
**end**

**Algorithm 1**: Modification of $\phi$ into a partial recursive prefix function

Intuitively, Algorithm 1 defines $\phi'$ only on those inputs $p$ such that the Turing machine $T$ halts first on $p$ in a dovetailed computation of all binary suffixes of $p$.

If $\phi\colon \{0,1\}^* \to \mathbb{N} \cup \{0\}$ is a partial recursive function and some distinct strings $xy, xz$ are in its domain then $xy$ and $xz$ cannot both be in the domain of $\phi'$, for $\phi$ halts first either on $xy_1$ or $xz_1$ in the computation of Algorithm 1. Hence the partial recursive function $\phi'$ determined by Algorithm 1 is indeed a partial recursive prefix function.

For any partial recursive function $f$ and any $x$ in its domain let $A_1(f, x)$ denote the result of the application of Algorithm 1 to $f$ and $x$. Let $T_1, T_2, ...$ be the effective enumeration of Turing machines constructed in Section 3 and let $\phi_1, \phi_2, ...$ be the corresponding enumeration of partial recursive functions, i.e. so that $f_{T_i} = \phi_i$.

For each $i \in \mathbb{N}$ and each $x$ in the domain of $\phi$ define $\phi'_i(x) = A_1(\phi, x)$ if the algorithm eventually finishes, and keep $\phi'_i(x)$ undefined otherwise. Since $\phi'_i$ is partial recursive there is a Turing machine $T'_i$ for which $\phi'_i = f_{T'_i}$. The machine $T'_i$ is a prefix machine since $\phi'_i$ is a partial recursive prefix function and $L(T'_i) = \mathrm{dom}(\phi'_i)$. If $T_i$ was already a prefix machine then $\phi = \phi'$ and $T_i = T'_i$. Consequently this procedure yields an effective enumeration $T'_1, T'_2, ...$ of all and only prefix machines. □

The technique illustrated in Algorithm 1 is called *dovetailing* a computation, and it was initially inspired by Cantor's diagonalization argument.

The following will be useful later and is a sort of consistency check for our definitions.

**Proposition 4.9.** *A partial recursive function $\rho\colon \{0,1\}^* \to \mathbb{N}$ is a partial recursive prefix function if and only if there is a prefix machine $T_\rho$ such that $\rho = f_{T_\rho}$.*

*Proof.* Since $\rho$ is a partial recursive prefix function there is a Turing machine $T$ such that $f_T = \rho$ and $\mathrm{dom}(\rho) = L(T)$. Thus $\rho$ is a partial recursive prefix function iff $T$ is a prefix machine. □

## 5. Prefix complexity and universal semimeasure

We first introduce the asymtoptically unique definitions of Kolmogorov complexity and prefix complexity, and we then construct an effective enumeration of lower semicomputable discrete semimeasures in order to construct a universal lowersemicomputable semimeasure $m(x)$. In Section 7.2 we will show that, under certain conditions, Bayesian inference using this semimeasure $m(x)$ as a prior selects the same hypothesis as ideal MDL.

### 5.1. **Kolmogorov complexity.**

**Definition 5.1.** Define the function $\langle -,-\rangle\colon \mathbb{N}\cup\{0\}\times\mathbb{N}\cup\{0\} \to \{0,1\}^*$ by $\langle n,m\rangle = 1^{l(b(n))}0b(n)b(m)$.

The function $\langle -,-\rangle$ is in fact a total recursive bijection.

**Definition 5.2.** For any $x, y \in \mathbb{N}$ and any partial recursive function $\phi\colon \{0,1\}^* \to \mathbb{N}$ we call a number $p \in \mathbb{N}$ such that $\phi(\langle y,p\rangle) = x$ a *$\phi$-description of $x$* given $y$.

**Definition 5.3.** For $\phi\colon \{0,1\}^* \to \mathbb{N}$ a partial recursive function, for any $x, y \in \mathbb{N}$ we define the *$\phi$-complexity of $x$ given $y$* to be

$$C_\phi(x|y) = \min_{p\in\mathbb{N}}\{l(b(p)) : \phi(\langle y,p\rangle) = x\}.$$

We leave $C_\phi(x|y)$ undefined if $x$ is not in the image of $\phi(\langle y,-\rangle)$.

In other words the $\phi$-complexity of $x$ given $y$ is the length of the shortest $\phi$-description of $x$ given $y$.

**Definition 5.4.** Fix a Turing machine $V$ which, given the input $\langle y,\langle n,p\rangle\rangle \in \{0,1\}^*$, enumerates the first $n$ Turing machines $T_1, T_2, ..., T_n$ from Section 3 and then computes $T_n(\langle y,p\rangle)$. Let $\Phi = f_V$ be the partial recursive function computed by $V$.

For any $x, y \in \mathbb{N}$ we define the *conditional Kolmogorov complexity of $x$ given $y$* by

$$C(x|y) = C_\Phi(x|y) = \min_{n,p\in\mathbb{N}}\{l(b(\langle n,p\rangle))\colon \Phi(\langle y,\langle n,p\rangle\rangle) = T_n(\langle y,p\rangle) = x\},$$

and we define the *Kolmogorov complexity of $x$* to be

$$C(x) = C(x|\epsilon),$$

where $\epsilon$ is the empty string.

**Definition 5.5.** Define the function $f\colon \mathbb{N}\cup\{0\}\times\{0,1\}^* \to \{0,1\}^*$ by

$$f(n,x) = \left\langle \pi_1(\langle -,-\rangle^{-1}(x)), \langle n, \pi_2(\langle -,-\rangle^{-1}(x))\rangle\right\rangle,$$

where $\pi_i$ denotes projection mapping onto the $i$-th coordinate.

The Turing machine $V$ is in fact a universal Turing machine since it satisfies Definition 3.7 with respect to the total recursive bijection $f$ from Definition 5.5. Such a $V$ may be constructed with respect to the effective enumeration $T_1, T_2, ....$

The following proposition justifies the terminology "the" Kolmogorov complexity by showing the asymptotic uniqueness of the quantity $C(x|y)$ relative to a fixed $\Psi$.

**Proposition 5.6.** *For any partial recursive function $\phi\colon \{0,1\}^* \to \mathbb{N}$ there exists a constant $c_\phi \geq 0$ such that*

$$C(x|y) \leq C_\phi(x|y) + c_\phi$$

*for all $x$ in the image of $\phi(\langle y, -\rangle)\colon \mathbb{N} \to \mathbb{N}$.*

*Proof.* Let $p \in \mathbb{N}$ be the shortest $\phi$-description of $x$ given $y$, so that $\phi(\langle y, p\rangle) = x$ and $l(p) = C_\phi(x|y)$. Since $\phi$ is a partial recursive function there exists some $j \in \mathbb{N}$ such that $\phi = \phi_j = f_{T_j}$, the partial recursive function associated to the $j$-th Turing machine in the fixed enumeration. By the definition of $V$

$$\phi(\langle y, p\rangle) = \phi_j(\langle y, p\rangle) = \Phi(\langle y, \langle j, p\rangle\rangle) = x,$$

so that $\langle j, p\rangle$ is a $\Phi$-description of $x$ given $y$. Hence

$$C(x|y) = C_\Phi(x|y) \leq l(\langle j, p\rangle) = 2l(b(j)) + 1 + l(b(p)) = C_\phi(x|y) + c_\phi$$

for all $x, y \in \mathbb{N}$, where $c_\phi = 2l(b(j)) + 1$. □

It is worth noting that the particular constant $c_\phi$ in the above proof depends on both a particular choice of function $\langle -, -\rangle\colon \mathbb{N} \times \mathbb{N} \to \{0,1\}^*$ and a particular Turing machine $V$ satisfying the condition in Definition 5.5.

Since any such Turing machine $V$ is a Turing machine (so that the function $\Phi$ eventually shows up in the list $\phi_1, \phi_2, ...$) and since

$$\lim_{n\to\infty} C(n) = \infty$$

this proposition shows that the function $C(x)$ is asymptotically independent of the choice of universal Turing machine.

## 5.2. **Prefix complexity.**

**Definition 5.7.** Let $\psi\colon \{0,1\}^* \to \mathbb{N}$ be a partial recursive prefix function. We define the *prefix $\psi$-complexity of $x$ given $y$* to be $K_\psi(x|y) = C_\psi(x|y)$.

**Definition 5.8.** Fix a Turing machine $U$ for which $U(\langle y, \langle n, p\rangle\rangle) = M_n(\langle y, p\rangle)$ for $M_n$ the $n$-th prefix machine. Such a machine $U$ is known to exist since (as in Section 4.3) prefix machines are effectively enumerable and $\langle -, -\rangle$ is total recursive. Pending the following Lemma, we call $U$ the *reference universal prefix machine.*

**Lemma 5.9.** *The Turing machine $U$ is a universal prefix machine.*

*Proof.* Recall the definition of $\langle -, -\rangle$ just above Section 5.1.

Suppose that $\langle y, \langle n, p\rangle\rangle$ and $\langle y, \langle n, p'\rangle\rangle$ are both in the domain of $U$ for $b(p)$ a prefix of $b(p')$. Notice that

$$\langle y, \langle n, p\rangle\rangle = 1^{l(b(y))}0b(y)1^{l(b(n))}0b(n)b(p).$$

Now note that $U(\langle y, \langle n, p\rangle\rangle = M_n(\langle y, p\rangle)$ and $U(\langle y, \langle n, p'\rangle\rangle) = M_n(\langle y, p'\rangle)$ cannot both be defined, since $\langle y, p\rangle$ is a prefix of $\langle y, p'\rangle$, and $M_n$ is a prefix machine. Hence $U$ is a prefix machine.

For any prefix machine $M$ there exists some number $n \in \mathbb{N}$ such that $M$ is the $n$-th prefix machine in the enumeration of Section 4.3. Since $\langle -, - \rangle$ is a bijection for any $x \in \{0,1\}^*$ there exist unique $y, p \in \mathbb{N}$ such that $\langle y, p \rangle = x$. Hence

$$U(f(n,x)) = U(f(n,\langle y,p\rangle)) = U(\langle y, \langle n,p\rangle\rangle) = T'_n(\langle y,p\rangle) = M(x)$$

for all $x \in \{0,1\}^*$, where $f$ is as in Definition 5.5. Therefore $U$ is a universal prefix machine. □

**Definition 5.10.** Let $\Psi$ be the partial recursive prefix function computed by $U$. For any $x, y \in \mathbb{N}$ we define the *conditional prefix complexity of $x$ given $y$* to be

$$K(x|y) = K_\Psi(x|y),$$

and for each $x \in \mathbb{N}$ we define the *prefix complexity of $x$* by

$$K(x) = K(x|\epsilon).$$

The following is proved analogously to Proposition 5.6.

**Theorem 5.11.** *If $\psi$ is a partial recursive prefix function then there exists some $k_\psi \geq 0$ such that $K(x|y) \leq K_\psi(x|y) + k_\psi$ for all $x$ in the image of $\psi(\langle y, - \rangle)$.*

For a discussion of the relation between the functions $K$ and $C$ see section 3.1 of [1].

5.3. **Universal lower semicomputable discrete semimeasure.** In this section we use the notation $f(x) < \infty$ to denote the fact that the partial function $f$ is defined on $x$.

**Definition 5.12.** We call a partial recursive function $f\colon \mathbb{N} \to \mathbb{R}$ *lower semicomputable* if there exists a partial recursive function $\phi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ satisfying the conditions

$$\phi(x,k) < \infty \text{ only when } \phi(x,1), ..., \phi(x,k-1) < \infty, \tag{5.13}$$

$$\phi(x,k+1) \geq \phi(x,k), \text{ and} \tag{5.14}$$

$$\lim_{k\to\infty} \phi(x,k) = f(x) \tag{5.15}$$

for all $x, k \in \mathbb{N}$. We call $f$ *upper semicomputable* if $-f$ is lower semicomputable. A partial recursive function $f\colon \mathbb{N} \to \mathbb{R}$ is called *recursive* if it is both upper and lower semicomputable.

**Lemma 5.16.** *There exists an effective enumeration $g_1, g_2, ...$ of lower-semicomputable functions $N \to \mathbb{R}$.*

*Proof.* We will prove the lemma by defining two algorithms and then applying these algorithms to an effective enumeration $h_1, h_2, ...$ of real-valued partial recursive functions. For any partial recursive function $h\colon \mathbb{N} \to \mathbb{R}$ we already have a partial recursive function $\phi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ with $\lim_{n\to\infty} \phi(x,n) = h(x)$ but in general the conditions Equation 5.13 and Equation 5.14 may not be satisfied by $\phi$. Consider the following algorithms.

**Data**: A partial recursive function $\phi\colon \mathbb{N}\times\mathbb{N}\to\mathbb{Q}$
**Result**: A partial recursive function $\phi'\colon \mathbb{N}\times\mathbb{N}\to\mathbb{Q}$ that satisfies Equation 5.13 for all $x,k\in\mathbb{N}$
**begin**
  Let $M$ be a Turing machine for which $f^2_M=\phi$
  **for** $x$ in the domain of $\phi$ **do**
    **for** $i=1,2,3,...$ **do**
      **for** $j=1,...,i$ **do**
        Compute a step of $M$ on $\langle x,j\rangle$
        **if** $M$ has halted on $\langle x,j\rangle$ **then**
          **if** $M$ has halted on $\langle x,1\rangle,...,\langle x,j-1\rangle$ **then**
            Define $\phi'(x,j)=M(\langle x,j\rangle)$
          **end**
        **end**
      **end**
    **end**
  **end**
**end**

**Algorithm 2**: Modifying $\phi$ so that it satisfies Equation 5.13

**Data**: A partial recursive function $\phi'\colon \mathbb{N}\times\mathbb{N}\to\mathbb{Q}$ that satisfies Equation 5.13 for all $x,k\in\mathbb{N}$
**Result**: A partial recursive function $\rho\colon \mathbb{N}\times\mathbb{N}\to\mathbb{Q}$ that satisfies Equation 5.13 and Equation 5.14 for all $x,k\in\mathbb{N}$
**begin**
  Let $T$ be a Turing machine for which $f^2_T=\phi'$
  **for** $x$ in the domain of $\phi'$ **do**
    Define $\rho(x,1)=\phi'(x,1)$
    Let $k:=2$
    **for** $i=1,2,...$ **do**
      **for** $j=2,...,i+1$ **do**
        Compute a step of $T$ on $\langle x,j\rangle$
        **if** $T$ has halted on $\langle x,j\rangle$ **then**
          **if** $b^{-1}(T(\langle x,j\rangle))\geq\rho(x,l)$ *for* $l=1,...,k-1$ **then**
            Define $\rho(x,k):=\phi'(x,j)$
            Let $k:=k+1$
          **end**
        **end**
      **end**
    **end**
  **end**
**end**

**Algorithm 3**: Modifying $\phi'$ so that it satisfies Equation 5.13 and Equation 5.14

From the effective enumeration of Turing machines in Section 2 and Definition 3.10 we obtain an effective enumeration $h_1,h_2,...$ of partial recursive functions $\mathbb{N}\to\mathbb{R}$. For each $i\in\mathbb{N}$ let $g_i$ be the rational-valued partial recursive function from Definition 3.10, namely for which $\lim_{n\to\infty}g_i(x,n)=h_i(x)$. Define the function $g'_i$

to be the result of the application of Algorithm 2 to the function $g_i$, and define $\rho_i$ to be the result of the application of Algorithm 3 to the function $g_i'$. Now define

$$H_i(x) = \lim_{n\to\infty} \rho(x, n)$$

if the limit exists, and keep $H_i(x)$ undefined otherwise. Then each function $H_i$ is lower semicomputable, and if $h_i$ was already lower semicomputable then $h_i = H_i$. □

**Definition 5.17.** A *discrete semimeasure* is a partial function $P\colon \mathbb{N} \to \mathbb{R}$ such that

$$\sum_{x\in\mathrm{dom}(P)} P(x) \leq 1.$$

A *discrete probability measure* is a function $P\colon \mathbb{N} \to \mathbb{R}$ such that $\sum_{x\in\mathbb{N}} P(x) = 1$.

Via the effective bijection $b$ from Section 2 we could equivalently define a discrete semimeasure as a partial function $P\colon \{0,1\}^* \to \mathbb{R}$ satisfying the same inequality. Throughout this paper the terms "semimeasure" and "probability measure" will always mean discrete semimeasure and discrete probability measure, respectively.

**Proposition 5.18.** *There exists an effective enumeration $P_1, P_2, ...$ of lower semicomputable semimeasures.*

*Proof.* We will prove the proposition by defining an algorithm to modify any lower semicomputable function $g$ into a lower semicomputable semimeasure $P$.

For any lower semicomputable function $g$ there is a partial recursive function $\phi\colon \mathbb{N}\times\mathbb{N} \to \mathbb{Q}$ satisfying Equation 5.13 and Equation 5.14 and for which $\lim_{n\to\infty} \phi(x, n) = g(x)$ for all $x$ in the domain of $g$.

Consider the following algorithm.

**Data**: A partial recursive function $\rho\colon N \times \mathbb{N} \to \mathbb{Q}$ satisfying Equation 5.13 and Equation 5.14
**Result**: A lower semicomputable semimeasure $P\colon \mathbb{N} \to [0, 1]$
**begin**
  Let $P(x) := 0$ for all $x \in \mathbb{N}$
  Let $T$ be a Turing machine for which $f_T^2 = \rho$
  **for** $i = 1, 2, ...$ **do**
    **for** $j = 1, ..., i$ **do**
      Compute a step of $T$ on $\langle j, i + j - 1\rangle$
      **if** $T$ has halted on $\langle m, j\rangle$ for all $m \leq j$ **then**
        **if** $\sum_{l=1}^{j} \rho(l, j) \leq 1$ **then**
          Define $P(1) := \rho(1, j), ..., P(j) := \rho(j, j)$
        **else**
          Quit the algorithm
        **end**
      **end**
    **end**
  **end**
**end**

**Algorithm 4**: Modifying $\phi$ so that it is a semimeasure

Now let $g_1, g_2, ...$ be the effective enumeration of lower semicomputable functions from Lemma 5.8. For each $i \in \mathbb{N}$ let $\rho_i\colon \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ be a function satisfying

Equation 5.13 and Equation 5.14 such that $\lim_{n\to\infty} \rho_i(x,n) = g_i(x)$. Define the function $P_i$ to be the result of the application of Algorithm 4 to the function $\rho_i$. Then each function $P_i$ is a lower semicomputable semimeasure, and if $g_i$ was already a semimeasure then $g_i = P_i$. □

**Definition 5.19.** Relative to the enumeration $P_1, P_2, ...$ of lower semicomputable semimeasures from Proposition 5.18 we define the function $m(x)$ to be

$$m(x) = \sum_{j\in\mathbb{N}} 2^{-K(j)} P_j(x).$$

**Theorem 5.20.** *The function $m(x)$ is a lower semicomputable semimeasure. Among lower semicomputable semimeasures the function $m(x)$ is universal, in the sense that for any other lower semicomputable semimeasure $P$ there exists some constant $c_P > 0$ such that $c_P \cdot m(x) \geq P(x)$ for all $x$ in the domain of $P$.*

*Proof.* One readily verifies that $m(x)$ is a semimeasure since

$$\sum_{x\in\{0,1\}^*} m(x) = \sum_{x\in\{0,1\}^*} \sum_{i\in\mathbb{N}} 2^{-K(i)} P_i(x)$$

$$= \sum_{i\in\mathbb{N}} 2^{-K(i)} \sum_{x\in\{0,1\}^*} P_i(x) \leq \sum_{y\in\{0,1\}^*} 2^{-K(y)} \leq 1.$$

where the final inequality follows from Kraft's inequality (since $K\colon \{0,1\}^* \to \mathbb{N}$ is a prefix code) and the penultimate inequality follows from the fact that each $P_i$ is a semimeasure.

For each lower semicomputable semimeasure $P_j$ let $\rho_j\colon \mathbb{N}\times\mathbb{N} \to \mathbb{Q}$ be a partial recursive function satisfying Equations 5.13, 5.14, and 5.15. Now for each $x$ in the domain of $P_j$ and each $k \in \mathbb{N}$ define

$$\rho(x,k) = \sum_{j\in\mathbb{N}} 2^{-K(j)} \phi_j(x).$$

Observe that $\rho$ satisfies Equation 5.13 and Equation 5.14 and that

$$\lim_{n\to\infty} \rho(x,n) = \sum_{j\in\mathbb{N}} 2^{-K(j)} P_j(x) = m(x).$$

Hence it follows from the fact that $K(x)$ is lower semicomputable that the semimeasure $m(x)$ is lower semicomputable.

If $P$ is a lower semicomputable semimeasure then $P = P_j$ for some $j \in \mathbb{N}$, and thus $m(x) \geq 2^{-K(j)} \cdot P_j(x)$, so that setting $c_p = 2^{K(j)}$ yields

$$c_p \cdot m(x) \geq P_j(x) = P(x).$$

□

One should be wary that theorems above and below depend upon particular enumerations of Turing machines. By contrast, it is not true that there exists a recursive semimeasure which is universal among the class of all recursive semimeasures; c.f. Lemma 4.3.1 in [2].

**Proposition 5.21.** *If $P$ is a semimeasure then there exists a prefix code $E\colon \mathbb{N} \to \{0,1\}^*$ such that $l(E(x)) < \log\left(\frac{1}{P(x)}\right) + 2$ for all $x \in \mathbb{N}$.*

*Proof.* Define adjacent intervals $I(1), I(2), ...$ of respective lengths $P(1), P(2), ...$ such that the left edge of $I(1)$ is at 0, and the left edge of $I_{j+1}$ is the right edge of $I_j$ for all $j \in \mathbb{N}$. For each $x \in \mathbb{N}$ let $E(x)$ be the binary string corresponding to the largest binary interval $\Gamma_{E(x)} \subset I$. By Lemma 4.2 we know that $P(x)$ is covered by at most four binary intervals of length $l(E(x))$, so

$$|I_x| = P(x) \leq 4 \cdot |\Gamma_{E(x)}| = 4 \cdot 2^{-l(E(x))},$$

from which we obtain

$$l(E(x)) \leq \log\left(\frac{1}{P(x)}\right) + 2.$$

□

5.4. **Coding theorem.**

**Definition 5.22.** Relative to the reference universal prefix machine $U$ defined in Section 5.2 we define the *universal a priori probability of* $x$ by

$$Q_U(x) = \sum_{U(p)=x} 2^{-l(p)}.$$

Intuitively the quantity $Q_U(x)$ corresponds to the relative frequency of programs $p \in \{0,1\}^*$ on which the reference universal prefix machine $U$ outputs the string $x$ and halts.

**Theorem 5.23.** *(Coding theorem) There exists a constant $c > 0$ such that*

$$\log\left(\frac{1}{m(x)}\right) = \log\left(\frac{1}{Q_U(x)}\right) = K(x)$$

*up to the constant $c$ for all $x \in \{0,1\}^*$.*

*Proof.* Since $U$ is a prefix machine for all $p \in \{0,1\}^*$ the association $p \mapsto x$ such that $U(p) = x$ defines a prefix code, so we have from Kraft's inequality that

$$\sum_{x \in \{0,1\}^*} Q_U(x) = \sum_{x \in \{0,1\}^*} \sum_{U(p)=x} 2^{-l(p)} \leq 1.$$

By dovetailing the computation of $U$ on the enumeration of all programs (i.e. strings in $\{0,1\}^*$) as in the proof of Proposition 5.18 we see that $Q_U(x)$ is lower semicomputable. From Theorem 5.20 there is a constant $c_1 > 0$ such that $c_1 m(x) \geq Q_U(x)$ for all $x \in \{0,1\}^*$.

Since $K(x) = \min_{p \in \{0,1\}^*}\{l(p) \colon U(p) = x\}$ we have that $2^{-K(x)} \leq Q_U(x)$, and consequently

$$\log\left(\frac{1}{m(x)}\right) \leq \log\left(\frac{1}{Q_U(x)}\right) + \log(c_1) \leq K(x) + \log(c_1).$$

We now construct a partial recursive prefix function $\gamma$ such that $K_\gamma(x) \leq \log\left(\frac{1}{m(x)}\right) + 3$, in a fashion similar to the proof of Proposition 5.21.

Let $\phi \colon \{0,1\}^* \times \mathbb{N} \to \mathbb{Q}$ be a partial recursive function such that for all $x \in \{0,1\}^*$, $\lim_{k \to \infty} \phi(x,k) = m(x)$ and $\phi(x, k+1) \geq \phi(x,k)$ for all $k \in \mathbb{N}$, which is known to exist since $m(x)$ is lower semicomputable.

For all $x, k$ define the function

$$\psi(x,k) = \max_{i \in \mathbb{N}}\{2^{-i} \colon 2^{-i} \leq \phi(x,k) < 2^{-i+1} \text{ and } \phi(x,j) < 2^{-i}, \forall\, j < k\}$$

and let $\psi(x,k)=0$ if no such bound exists. It follows that $\psi$ is partial recursive since $\phi$ is partial recursive.

We will now modify the function $\psi$ into a function $\psi'$ which enumerates the range of $\psi$ without repetition for each $x$. More precisely, we construct $\psi'$ to satisfy $\psi'(x,n)\neq\psi'(x,m)$ for any $n\neq m$.

First define $\psi'(x,1):=\psi(x,1)$, and for each $i\geq 2$ let

$$k_i=\min_{k\in\mathbb{N}}\{k\colon \psi(x,k)\neq\psi'(x,j),\,\forall\, j<i\}.$$

Now put $\psi'(x,l):=\psi(x,k_l)$ for all $l\geq 2$. The function $\psi'$ thus defined is still partial recursive.

Note that for any $x$ there is a maximum $i_x\in\mathbb{N}$ for which $2^{-i}\leq m(x)<2^{-i+1}$, from which it follows that $\sum_k\psi'(x,k)\leq 2m(x)$. From this we deduce

$$\sum_{x,k}\psi'(x,k)=\sum_x\sum_k\psi'(x,k)\leq\sum_x 2m(x)\leq 2. \tag{5.24}$$

We will now define an algorithm to construct consecutive half-open intervals contained in $[0,1)$ which we will then use to define a prefix code.

**Data**: A partial recursive function $\psi\colon\{0,1\}^*\times\mathbb{N}\to\mathbb{Q}$ satisfying inequality Equation 5.24
**Result**: A collection of consecutive intervals $\{J_{x,k}\}$ contained in $[0,1)$ such that $J_{x,k}$ is of length $\frac{\psi(x,k)}{2}$

```
begin
    Fix a Turing machine M for which f_M^2 = ψ
    Let z := 0
    for i = 1, 2, ... do
        for j = 1, ..., i do
            for k = 1, ..., j do
                if M has not halted on ⟨b(j), k⟩ then
                    Compute a step of M on ⟨b(j), k⟩
                    if M has halted on ⟨b(j), k⟩ then
                        Define J_{b(j),k} = [z, ψ(b(j),k)/2 + z)
                        Let z := z + |J_{b(j),k}|
                    end
                end
            end
        end
    end
end
```

**Algorithm 5**: Construction of intervals for prefix code

Denote by $\{I_{x,k}\}$ the family of consecutive intervals in $[0,1)$ obtained by applying Algorithm 5 to the partial recursive function $\psi'$. Now define a partial function $\gamma\colon\{0,1\}^*\to\{0,1\}^*$ by letting $\gamma(p)=x$ if there exists some $k\in\mathbb{N}$ such that $\Gamma_p$ is the leftmost of the longest binary intervals contained in $I_{x,k}$. Since the intervals $\{I_{x,k}\}$ are disjoint $\gamma$ is a prefix function, and $\gamma$ is partial recursive since Algorithm 5 may be run on a Turing machine. Hence $\gamma$ is a partial recursive prefix function.

For each $x \in \{0,1\}^*$ we know from the definition of $\psi'$ above equation Equation 5.24 that there is some $s_x \in \mathbb{N}$ such that $\psi'(x, s_x) > \frac{m(x)}{2}$, namely the unique $s_x \in \mathbb{N}$ for which $\psi'(x, s_x) = 2^{-i_x}$.

Any interval in $[0,1)$ of length $2^{-N}$ contains a binary interval of length at least $2^{-N-1}$. Since the length of any nonempty $I_{x,k}$ is a negative exponent of 2 we have that there is a binary interval $\Gamma_{p_x} \subset I_{x,s_x}$ for which

$$|\Gamma_{p_x}| = 2^{-l(p_x)} \geq \frac{1}{2}|I_{x,s_x}| = \frac{\psi'(x, s_x)}{4} > \frac{m(x)}{8},$$

from which one readily obtains $l(p_x) \leq \log\left(\frac{1}{m(x)}\right) + 3$.

Since $p_x$ is a program on which the partial recursive prefix function $\gamma$ computes $x$, we have that $K(x) = K_\Psi(x) \leq K_\gamma(x) + c_3 \leq l(p_x) + c_4$ for some $c_3, c_4 \geq 0$ by Theorem 5.11. Consequently, letting $c_5 = c_2 + c_4$ we have

$$K(x) \leq l(p_x) + c_4 \leq \log\left(\frac{1}{m(x)}\right) + c_5$$

for all $x \in \{0,1\}^*$.

Putting this all together we have that the three functions $\log\left(\frac{1}{m(x)}\right)$, $\log\left(\frac{1}{Q_U(x)}\right)$, and $K(x)$ are equal up to the constant $\max\{c_5, \log(c_1)\}$ for all $x \in \{0,1\}^*$. □

For convenience we state the conditional version of this Theorem; its proof is virtually identical. By defining $P_j(x|y) = P_j(\langle x, y\rangle)$ we may define

$$m(x|y) = \sum_{j \in \mathbb{N}} 2^{-K(j)} P_j(x|y).$$

**Theorem 5.25.** *For all strings $y \in \{0,1\}^*$ we have that*

$$\log\left(\frac{1}{m(x|y)}\right) = K(x|y)$$

*up to a fixed constant $c > 0$ independent of $x$ and $y$.*

## 6. Randomness tests

In this section we introduce the universal sum $P$-test — for $P$ a recursive probability measure — as a rigorous notion of randomness. We proceed to show that with the universal distribution $m(x)$ we may construct a very natural universal sum $P$-test. This allows us to formulate a very intuitive condition under which conclusions of Bayesian inference (using $m(x)$ as a prior) and ideal MDL coincide for particular data samples.

### 6.1. Sum $P$-tests.

**Definition 6.1.** Let $P\colon \mathbb{N} \to \mathbb{R}$ be a recursive probability measure. A *sum $P$-test* is a total function $\delta\colon \mathbb{N} \to \mathbb{N}$ such that

- $\delta$ is lower semicomputable, and
- $\sum_{x \in \mathbb{N}} P(x) 2^{\delta(x)} \leq 1$.

The intuition here is that the sum $P$-test $\delta$ measures some "property" or "regularity" of $\mathbb{N}$ in the sense that larger values $\delta(n)$ indicate numbers $n$ that possess this property or regularity (e.g. number of 1's in the string $b(n)$) to a larger extent, and the inequality ensures that only a few numbers strongly exhibit this property (i.e. have large $\delta(n)$). The inequality ensures that elements which possess this property to larger degrees are increasingly rare.

**Definition 6.2.** Let $P$ be a recursive probability measure. A *universal sum $P$-test* is a sum $P$-test $\delta_P \colon \mathbb{N} \to \mathbb{N}$ such that for any sum $P$-test $\delta \colon \mathbb{N} \to \mathbb{N}$ there exists some $c_\delta > 0$ such that

$$\delta_P(x) \geq \delta(x) - c_\delta$$

for all $x \in \mathbb{N}$.

Roughly analogous to a universal partial recursive function a universal $P$-test detects every property that every other $P$-test measures, up to the constant $c_\delta$.

**Theorem 6.3.** *For any recursive probability measure $P$ the function $\kappa_P \colon \mathbb{N} \to \mathbb{N}$ defined by*

$$\kappa_P(x) = \log\left(\frac{m(x)}{P(x)}\right)$$

*is a universal sum $P$-test, where $m(x)$ is the universal lower semicomputable semimeasure constructed in Section 5.3.*

*Proof.* By the definition of $\kappa_P$ we obtain

$$\sum_{x \in \{0,1\}^*} P(x) 2^{\kappa_P(x)} = \sum_{x \in \{0,1\}^*} P(x) \frac{m(x)}{P(x)} \leq 1$$

since $m$ is a semimeasure, so $\kappa_P$ is a sum $P$-test.

Now suppose that $\delta \colon \{0,1\}^* \to \mathbb{N}$ is a sum $P$-test. Observe that $P(x)2^{\delta(x)}$ is necessarily a lower semicomputable semimeasure, and since $m(x)$ is the universal lower semicomputable semimeasure there is some constant $c > 0$ such that $cm(x) > P(x)2^{\delta(x)}$ and therefore

$$\kappa_P(x) = \log\left(\frac{m(x)}{P(x)}\right) \geq \delta(x) - \log(c)$$

for all $x \in \{0,1\}^*$, so $\kappa_0(-|P)$ is indeed a universal sum $P$-test. □

## 7. Hypothesis identification

We first discuss the notion of a two-part code and then formally introduce the principles of Bayesian inference and ideal MDL. With simplified assumptions we analyze the conditions under which these two principles select the same hypothesis.

In this section we will use the notation $f \overset{+}{<} g$ when there exists some nonnegative constant $c \in \mathbb{R}$ such that $f(x) < g(x) + c$ for all $x$ in the domains of $f$ and $g$, and $f \overset{+}{=} g$ if both $f \overset{+}{<} g$ and $g \overset{+}{<} f$.

7.1. **Two-part codes.** The intuition behind two-part codes is rather natural with respect to the computability theory discussed in this paper, especially when one is looking to describe some data $D$ (say, a string in $\{0,1\}^*$) which is regular, random, or anywhere in between.

In defining the complexities $C(x)$ and $K(x)$ we referred not only to a program $p \in \{0,1\}^*$ but also implicitly to a prefix machine $M_i$ which the reference universal machine simulates with the input $p$ to compute $x$; we can think of the prefix machine $M_i$ as representing some regularity, and of $p$ as representing $x$ with respect to that regularity.

In the ideal MDL approach we will be interested, given some data sample $D$, in the hypothesis $H$ which minimizes $K(D|H) + K(H)$. This is intuitively the length of the shortest two-part code for $D$, which first describes the hypothesis $H$ and then describes $D$ relative to $H$, namely by giving its index in an enumeration of the elements of $H$.

7.2. **Principles of hypothesis identification.** Consider the situation where the data is some string $D \in \{0,1\}^n$ and the class of contemplated hypotheses is $\mathcal{H} = \mathcal{P}(\{0,1\}^n)$. We already know how to define the complexity $K(D)$, but to use ideal MDL we must also define $K(H)$ and $K(D|H)$.

As in previous sections we fix a reference universal prefix machine $U$.

**Definition 7.1.** For any subset $H \subset \{0,1\}^n$ we define the prefix complexity of the hypothesis $K(H)$ to be the length of the shortest program $p$ for which $U(p) = \langle h_1, \langle h_2, \langle \cdots h_{k-1}, h_k \rangle \cdots \rangle$, where $H = \{h_1, ..., h_k\}$ and $h_1, ..., h_k$ are ordered lexicographically (i.e. $b^{-1}(h_i) \leq b^{-1}(h_j)$ for all $i < j$).

We define the conditional complexity $K(D|H)$ of the data given the hypothesis as the length of the shortest program on which $U(\langle p, \langle h_1, \langle h_2, \langle \cdots h_{k-1}, h_k \rangle \cdots \rangle \rangle) = x$, that is, such that $U$ computes $x$ given the enumeration of $H$.

For any $H \subset \{0,1\}^n$ we have that $K(D|H) \leq \log(d(H))$ (where $d$ denotes the cardinality, e.g. the number of strings in $H$) since we could always compute $x$ by giving its index in the enumeration of the elements of $H$.

**Definition 7.2.** For a data sample $D \in \{0,1\}^*$, a hypothesis class $\mathcal{H}$ and prior probabilities $\Pr(-)$ over the class $\mathcal{H}$ and $\Pr(-|H)$ over $\{0,1\}^*$ for each $H \in \mathcal{H}$, the principle of *Bayesian inference* selects the hypothesis

$$H_B = \arg\min_{H \in \mathcal{H}} \{-\log(P(H)) - \log(\Pr(D|H)\} = \arg\max_{H \in \mathcal{H}} \{P(H) \cdot \Pr(D|H)\},$$

which is derived from Equation 1.2. If more than one hypothesis achieves this minimum then we select the hypothesis with minimum complexity.

**Definition 7.3.** For a data sample $D \in \{0,1\}^*$ and a hypothesis class $\mathcal{H}$ the *ideal MDL* principle selects the hypothesis

$$H_M = \arg\min_{H \in \mathcal{H}} \{K(H) + K(D|H)\},$$

where the complexity $K(H)$ is defined to be the length of the first shortest program on which the reference universal prefix machine $U$ prints out the characteristic sequence of $H$ (which has its $i$-th term equal to 1 iff $b(i) \in H$). If multiple hypotheses achieve this minimum then we select the one with minimum complexity.

Since a description of $x$ relative to some hypothesis $H$ together with a description of $H$ is a description of $x$, we always have that $K(D) \stackrel{+}{<} K(D|H) + K(H)$. The idea of the Kolmogorov minimal sufficient statistic is to find the least complex hypothesis for which $K(D|H) + K(H)$ is close to $K(D)$.

**Definition 7.4.** For a data sample $D \in \{0,1\}^n$ and a hypothesis class $\mathcal{H} \subset \{H\colon H \subset \{0,1\}^n\}$ the *Kolmogorov minimal sufficient statistic* is the shortest program $H_S^*$ that prints out the characteristic sequence of

$$H_S = \arg\min_{H\in\mathcal{H}}\{K(H)\colon K(D|H) + \log(d(H)) \stackrel{+}{=} K(D)\}.$$

The above principle is well-defined since for any $D \in \{0,1\}^n$ one could always select the trivial hypothesis $H_D = \{D\}$ for which $K(H_D) \stackrel{+}{=} K(D)$ and $\log(d(H_D)) = 0$ so there will always be some hypothesis $H$ satisfying $K(D|H) + \log(d(H)) \stackrel{+}{=} K(D)$.

The following theorem addresses only the situations in which the contemplated hypotheses are finite collections of finite, binary strings.

**Theorem 7.5.** *Let $n$ be a large enough positive integer. Given some data sample $D \in \{0,1\}^n$ and the hypothesis class $\mathcal{H} = \{H\colon H \subseteq \{0,1\}^n\}$ the following three principles select the same hypothesis:*

- **a:** *Bayesian inference with $P(H) = m(H)$ and $-\log(Pr(D)) = \log(d(H))$;*
- **b:** *Kolmogorov minimal sufficient statistic;*
- **c:** *Ideal MDL.*

*Proof.* ([a] $\Leftrightarrow$ [b]). Note first that for any hypothesis $H \in \mathcal{H}$ we have $K(H) + K(D|H) \stackrel{+}{>} K(D)$ since the shortest description of $D$ is less than the description of $D$ in terms of $H$. Note also that $K(D|H) \stackrel{+}{<} \log(d(H))$ for any $H \in \mathcal{H}$ since we could always describe $D$ given $H$ with its index in an effective enumeration of the elements of $H$. Thus for any $H \in \mathcal{H}$ we have $K(H) + \log(d(H)) \stackrel{+}{>} K(D)$.

For ease of notation let

$$\mathcal{H}_B = \left\{H'\colon H' = \arg\min_{H\in\mathcal{H}}\{-\log(P(H)) - \log(\Pr(D|H))\}\right\},$$

so that $H_B = \arg\min_{H\in\mathcal{H}_B}\{K(H)\}$.

By Theorem 5.25 we have $-\log(m(H)) = K(H) + O(1)$ so that $-\log(m(H)) \stackrel{+}{=} K(H)$ for $n$ sufficiently large and therefore

$$H_B = \arg\min_{H\in\mathcal{H}}\{K(H)\colon K(H) + \log(d(H)) \stackrel{+}{=} K(D)\} = H_S$$

as required.

([b] $\Leftrightarrow$ [c]). For the hypothesis $H_S$ selected by the Kolmogorov minimal sufficient statistic we have $K(H_S) + \log(D(H)) \stackrel{+}{=} K(D)$ so that

$$K(H) + K(D|H) \stackrel{+}{>} K(D) \stackrel{+}{=} K(H_s) + K(D|H_s)$$

for any hypothesis $H \in \mathcal{H}$. Therefore, letting

$$\mathcal{H}_0 = \{H'\colon H' = \arg\min_{H\in\mathcal{H}}\{K(H) + K(D|H)\}\}$$

we have that

$$H_I = \arg\min_{H \in \mathcal{H}_0} \{K(H)\} = H_S.$$

□

The generalization of these findings to probablistic hypotheses can be found in [2].

## 8. Acknowledgments

I am very grateful for my mentor, Angela Wu, who has been a phenomenal resource throughout my attempts to learn and express this material. I would also like to thank Professor May for organizing this wonderful program.

## References

[1] Ming Li and Paul Vitanyi. An introduction to Kolmogorov complexity and its applications. Springer. 2008.
[2] Ming Li and Paul Vitanyi. Minimim description length induction, Bayesianism, and Kolmogorov complexity. IEEE: *Transactions on Information Theory*, vol. 46, no. 2, March 2000.
[3] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to automata theory, languages, and computation. Pearson. 2001.