

On the Computability of Finding Capacity-Achieving Codes

Angelos Gkekas, Nikos A. Mitsiou, *Graduate Member, IEEE*,
Ioannis Souldatos, and George K. Karagiannidis, *Fellow, IEEE*

Abstract—This work studies the problem of constructing capacity-achieving codes from an algorithmic perspective. Specifically, we prove that there exists a Turing machine which, given a discrete memoryless channel $p_{Y|X}$, a target rate R less than the channel capacity $C(p_{Y|X})$, and an error tolerance $\epsilon > 0$, outputs a block code $\mathcal{C}$ achieving a rate at least R and a maximum block error probability below ϵ . The machine operates in the general case where all transition probabilities of $p_{Y|X}$ are computable real numbers, and the parameters R and ϵ are rational. The proof builds on Shannon’s channel coding theorem and relies on an exhaustive search approach that systematically enumerates all codes of increasing block length until a valid code is found. This construction is formalized using the theory of recursive functions, yielding a μ -recursive function $\text{FindCode} : \mathbb{N}^3 \rightarrow \mathbb{N}$ that takes as input appropriate encodings of $p_{Y|X}$, R , and ϵ , and, whenever $R < C(p_{Y|X})$, outputs an encoding of a valid code. By Kleene’s normal form theorem, which establishes the computational equivalence between Turing machines and μ -recursive functions, we conclude that the problem is solvable by a Turing machine. This result can also be extended to the case where ϵ is a computable real number, while we further discuss an analogous generalization of our analysis when R is computable as well. We note that the assumptions that the probabilities of $p_{Y|X}$, as well as ϵ and R are computable real numbers cannot be further weakened, since computable reals constitute the largest subset of $\mathbb{R}$ representable by algorithmic means.

Index Terms—Capacity-achieving codes, Turing machines, recursive functions, channel coding theorem, discrete memoryless channels (DMCs).

I. INTRODUCTION

Capacity-achieving codes constitute a cornerstone of modern communication theory. They enable reliable information transmission over noisy environments by ensuring that the probability of decoding error can be made arbitrarily small, while incurring only a bounded increase in codeword length. Specifically, for any discrete memoryless channel (DMC) without feedback, characterized by the conditional distribution $p_{Y|X}$, there exists a fundamental limit $C(p_{Y|X})$, referred to as the channel capacity. Shannon’s channel coding theorem establishes that for any desired error tolerance $\epsilon > 0$ and any coding rate $R \in (0, C)$, there exists a coding scheme $\mathcal{C}$ that simultaneously achieves error probability smaller than ϵ and rate at least R . This seminal result, first established in

Shannon’s groundbreaking work [1], laid the foundation for the entire field of modern digital communications.

However, Shannon’s original proof did not provide a formal approach for designing such codes. Since then, a variety of explicit code constructions that approach or achieve capacity have been developed. Among the most widely studied are low-density parity-check (LDPC) codes [2], Reed–Solomon codes [3], turbo codes [4], and polar codes [5]. These codes have been extensively analyzed and successfully implemented in practical communication systems and standards, where they have demonstrated exceptional performance, underscoring the lasting importance of Shannon’s original result. Nonetheless, each of these constructions is typically tailored to specific families of channel models. Naturally, this raises the question of whether constructing capacity-achieving codes for arbitrary DMCs is feasible from a computability perspective.

Among the most well-known tools for studying computability are Turing machines and μ -recursive functions. Turing machines, introduced by Turing in his seminal work [6], constitute the earliest and most widely used formal model of computation, and are regarded as the foundation of theoretical computer science. The class of μ -recursive functions provides an alternative model, defined as a collection of partial functions from tuples over $\mathbb{N}$ to $\mathbb{N}$. A notable subclass is that of the primitive recursive functions, first introduced in Gödel’s proof of the incompleteness theorems [7]. Their first complete formulation was later developed in Kleene’s studies on recursion theory [8]. For a comprehensive exposition of μ -recursive functions, see [9]–[12]. These two models of computation are equivalent, in the sense that any computational task performed in one model can be simulated in the other. This equivalence is established by Kleene’s normal form theorem [9], which allows us to freely adopt either model in our analysis.

A. Literature Review

There are only a handful of relevant contributions that examine the computability of constructing channel codes for DMCs. First, in [13], the computability of the zero-error capacity for DMCs was studied using Turing machines and Kolmogorov oracles and it was proved that the zero-error capacity function is semi-computable in this setting, while [14] employed a probabilistic channel coding construction to obtain capacity-achieving linear codes for a broad class of channels. However, the most fundamental work on the computability of constructing channel codes is [15]. In [15], the following question was addressed

Angelos Gkekas, Nikos A. Mitsiou, and George K. Karagiannidis are with the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece (e-mails: gkekasaa@ece.auth.gr, nmitsiou@auth.gr, geokarag@auth.gr).

Ioannis Souldatos is with the Department of Mathematics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece (e-mail: souldatos@math.auth.gr).

Question 1. For a given computable family of channels $\mathcal{W}$ and an error tolerance $\epsilon \in (0, 1) \cap \mathbb{Q}$, does there exist a Turing machine $M_{\mathcal{W}, \epsilon}$ such that, for every DMC $p_{Y|X} \in \mathcal{W}$ and every blocklength n , the machine outputs a block code $\mathcal{C}_n$ of length n , whose maximum block error probability λ_n and rate R_n satisfy $\lambda_n \rightarrow 0$ and $R_n \rightarrow C(p_{Y|X})$ as $n \rightarrow \infty$?

A negative answer to this question was provided using tools from the theory of Turing machines, recursive functions and computable real analysis. Moreover, this result also implies that such an algorithmic construction remains impossible even when the optimality condition is dropped and codes only need to achieve a fraction of the capacity.

B. Motivation and Contribution

The negative answer to the problem formulation stated in [15] motivates the exploration of weaker assumptions for the formulation of the problem of constructing capacity-achieving codes, under which the problem is computable. Ideally, one is interested in finding the least weak assumptions for which this problem remains computable. As a first step in this direction, Question 2 arises

Question 2. Does there exist a Turing machine M such that, when given as input a DMC $p_{Y|X}$, a rate $R < C(p_{Y|X})$ and an error tolerance $\epsilon > 0$, it outputs a block code $\mathcal{C}$ with rate at least R and with maximum block error probability $\lambda_{\max} < \epsilon$?

We prove that the answer to this question is positive. Specifically, we show that the above problem is solvable by a Turing machine in the general setting where all parameters of the channel $p_{Y|X}$ are arbitrary computable real numbers, while the error tolerance ϵ and the rate R are rational. We also extend this result by allowing ϵ to be any computable real number, while we further provide an informal discussion of a similar extension of R to computable real numbers too. To prove this, first, we describe a solution to the capacity-achieving code construction problem via an exhaustive search algorithm that relies on appropriately predefined μ -recursive functions. We then formally construct a μ -recursive function that implements this algorithm. Finally, we invoke Kleene's normal form theorem to conclude that the problem is solvable by a Turing machine. We note that the intermediate step of converting the algorithm into a μ -recursive function ensures full rigor, since it removes ambiguities in the implementation of certain operations, such as the representation of computable reals or rationals of arbitrary precision and the arithmetic performed on them. By contrast, the final conversion from a μ -recursive function to a Turing machine is primarily aesthetic, as Turing machines are the standard formalism in which computability results are typically expressed.

Interestingly, the positive answer to Question 2 also implies the existence of a Turing machine that takes as input a DMC $p_{Y|X}$ and a parameter $k \in \mathbb{N}$, and computes a code $\mathcal{C}_k$ that achieves a rate of at least $C(p_{Y|X}) - \frac{1}{k}$ with a maximum block error probability below $\frac{1}{k}$. Hence, the sequence of codes $\{\mathcal{C}_k\}$ satisfies the desired asymptotic properties of the rate approaching the channel capacity and the error probability

tending to zero. The detailed construction of this Turing machine is discussed in Appendix A.

At first glance, this may seem to contradict the negative answer to Question 1 by [15]. However, this is not the case for two main reasons. First, the machine proven impossible in [15] takes as input a DMC $p_{Y|X}$ and a number n , and outputs a code $\mathcal{C}_n$ with blocklength exactly n . In contrast, the machine proposed in this work takes as input a DMC $p_{Y|X}$ and a number k , and produces a code $\mathcal{C}_k$ without any restriction on its blocklength. Second, [15] defines the general notion of *computable families* of DMCs and prove their results under the assumption that the DMC input space is one such family. In our case, we consider arbitrary DMCs of any dimension, but we require that all transition probabilities $p_{Y|X}$ are computable real numbers.

II. OUTLINE

The remainder of this paper is organized as follows. Section III introduces some key concepts and results from the theory of recursive functions. In Section IV, we present the model of computable real numbers used throughout the paper. Section V formulates the main problem and develops a recursive function that solves it. Finally, Section VI offers concluding remarks.

III. RECURSION FRAMEWORK

In this section, we introduce some concepts from the theory of recursive functions that are fundamental to our analysis. These include the classes of primitive recursive and μ -recursive functions, primitive recursive encodings, Turing machines, Kleene's normal form theorem, and a recursion framework for functions and relations of rational numbers. These concepts can be traced back to the works of Gödel [7], Turing [6], and Kleene [8], [16], [17]. For a modern exposition, we refer the reader to [10]–[12].

A. Primitive and μ -Recursion

We begin by introducing the concepts of primitive recursiveness, μ -recursiveness and primitive recursive encodings. As a first step, we define the notion of a *partial function*.

Definition 1 (Partial Function). A *partial function* $f : X \rightarrow Y$ is a function $f : S \rightarrow Y$ with $S \subseteq X$. The set S is called the *domain* of f and it is denoted by $\text{dom}(f)$. If $x \in S$, we write $f(x) \downarrow$ and if $x \in X \setminus S$ we write $f(x) \uparrow$ or $f(x) = \perp$. If $S = X$, then f is called a *total function*.

We will mainly work with partial functions of the form $f : \mathbb{N}^n \rightarrow \mathbb{N}$ for $n \in \mathbb{N}$. We now define the operations of *composition*, *primitive recursion* and *minimization* for such functions.

Definition 2 (Composition of Partial Functions). Let $f : \mathbb{N}^m \rightarrow \mathbb{N}$ and $g_1, g_2, \dots, g_m : \mathbb{N}^n \rightarrow \mathbb{N}$ be *partial functions*. The *composition* $h(\bar{x}) = f(g_1(\bar{x}), g_2(\bar{x}), \dots, g_m(\bar{x}))$ is a *partial function* $h : \mathbb{N}^n \rightarrow \mathbb{N}$ defined as:

$$h(\bar{x}) = \begin{cases} f(c_1, c_2, \dots, c_m), & \text{if } g_i(\bar{x}) = c_i \neq \perp, \forall i \\ & \text{and } f(c_1, c_2, \dots, c_m) \downarrow \\ \perp, & \text{otherwise} \end{cases} \quad (1)$$

Definition 3 (Primitive Recursion). Let $g : \mathbb{N}^n \rightarrow \mathbb{N}$ and $h : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ be partial functions. It can be shown that there exists a unique partial function $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ such that:

$$\begin{cases} f(0, \bar{x}) = g(\bar{x}), & \forall \bar{x} \in \mathbb{N}^n \\ f(y+1, \bar{x}) = h(f(y, \bar{x}), y, \bar{x}), & \forall y \in \mathbb{N}, \forall \bar{x} \in \mathbb{N}^n \end{cases} \quad (2)$$

The compositions in the above expression are interpreted in accordance with definition 2. This means that:

$$\begin{cases} f(0, \bar{x}) \downarrow \Leftrightarrow g(\bar{x}) \downarrow \\ f(y+1, \bar{x}) \downarrow \Leftrightarrow f(y, \bar{x}) = c \neq \perp \text{ and } h(c, y, \bar{x}) \downarrow \end{cases} \quad (3)$$

We say that this function f is defined by primitive recursion on h with base case g .

Definition 4 (Minimization). Let $g : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ be a partial function. Define $D_{x, \bar{y}} = \{i \in \mathbb{N} \mid i \geq x, g(i, \bar{y}) = 0 \text{ and } g(j, \bar{y}) \downarrow \text{ for all } j \text{ with } x \leq j \leq i\}$. The minimization of g is defined as the partial function $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ with $f(x, \bar{y}) = \mu i \geq x : (g(i, \bar{y}) = 0)$ where:

$$\mu i \geq x : (g(i, \bar{y}) = 0) = \begin{cases} \min D_{x, \bar{y}}, & \text{if } D_{x, \bar{y}} \neq \emptyset \\ \perp, & \text{otherwise} \end{cases} \quad (4)$$

If $x = 0$ we write for simplicity $\mu i : (g(i, \bar{y}) = 0)$ instead of $\mu i \geq 0 : (g(i, \bar{y}) = 0)$.

We now proceed to define the classes of *primitive recursive* and μ -recursive functions. To do so, we first introduce the set of *basic functions* from which these classes are constructed.

Definition 5 (Basic Functions). The set B of basic functions is the set that includes exactly the following functions:

- 1) For $n, k \in \mathbb{N}$ the constant functions $C_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$ with $C_k^n(\bar{x}) = k$
- 2) For $n \in \mathbb{N}$ and $0 \leq i < n$ the projections $P_i^n : \mathbb{N}^n \rightarrow \mathbb{N}$ with $P_i^n(x_0, x_1, \dots, x_{n-1}) = x_i$
- 3) The successor function $S(x) = x + 1$

Definition 6 (Primitive Recursive and μ -Recursive Functions). The class R_p of primitive recursive functions is the closure of B for composition and primitive recursion. The class R_μ of μ -recursive functions is the closure of B for composition, primitive recursion and minimization.

The notions of primitive and μ -recursiveness also extend to relations.

Definition 7 (Primitive Recursive and μ -Recursive Relations). The characteristic function of a relation $R \subseteq \mathbb{N}^n$ is the function $\chi_R : \mathbb{N}^n \rightarrow \mathbb{N}$ defined as:

$$\chi_R(\bar{x}) = \begin{cases} 1, & \text{if } R(\bar{x}) \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

A relation R is called *primitive recursive* (μ -recursive) iff $\chi_R \in R_p$ (iff $\chi_R \in R_\mu$).

Furthermore, we introduce the concept of *primitive recursive encodings*.

Definition 8 (Primitive Recursive Encoding). We denote by $\mathbb{N}^* = \bigcup_{n \in \mathbb{N}} \mathbb{N}^n$ the set of all finite sequences of natural numbers, including the empty sequence ε . A function $\langle \cdot \rangle : \mathbb{N}^* \rightarrow \mathbb{N}$ is called a *primitive recursive encoding* iff it satisfies the following conditions:

- 1) $\langle \cdot \rangle$ is injective
- 2) The relation $\text{seq} \subseteq \mathbb{N}$ defined by:

$$\text{seq}(u) \Leftrightarrow \exists \bar{x} \in \mathbb{N}^* : \langle \bar{x} \rangle = u \quad (6)$$

is primitive recursive

- 3) The functions $F_n : \mathbb{N}^n \rightarrow \mathbb{N}$ defined by:

$$F_n(\bar{x}) = \langle \bar{x} \rangle \quad (7)$$

are primitive recursive for all $n \in \mathbb{N}$

- 4) $\langle x_0, x_1, \dots, x_{n-1} \rangle > x_i$ for all $(x_0, x_1, \dots, x_{n-1}) \in \mathbb{N}^*$ and $0 \leq i < n$
- 5) There exist primitive recursive functions $\text{lh} : \mathbb{N} \rightarrow \mathbb{N}$ and $\text{app}, \text{proj} : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for all $(x_0, x_1, \dots, x_{n-1}) \in \mathbb{N}^*$, $0 \leq i < n$ and $y \in \mathbb{N}$:

- $\text{lh}(\langle x_0, x_1, \dots, x_{n-1} \rangle) = n$
- $\text{app}(\langle x_0, x_1, \dots, x_{n-1} \rangle, y) = \langle x_0, x_1, \dots, x_{n-1}, y \rangle$
- $\text{proj}(\langle x_0, x_1, \dots, x_{n-1} \rangle, i) = x_i$

In general, we are only concerned with the values of the functions lh , app and proj when their arguments are of the form specified in condition (5) of the preceding definition. Although these functions are also well-defined for inputs not of this form, their values in these cases are not relevant.

For a given primitive recursive encoding we will use the notation $(u)_{i_1, i_2, \dots, i_k}$ to denote nested application of the function proj as:

$$(u)_{i_1, i_2, \dots, i_k} = \text{proj}(\dots \text{proj}(\text{proj}(u, i_1), i_2) \dots, i_k) \quad (8)$$

An example of a primitive recursive encoding is the *classical encoding* defined by:

$$\begin{cases} \langle \varepsilon \rangle = 1 \\ \langle x_0, x_1, \dots, x_{n-1} \rangle = p_0^{x_0+1} \cdot p_1^{x_1+1} \cdot \dots \cdot p_{n-1}^{x_{n-1}+1} \end{cases} \quad (9)$$

where p_i denotes the i -th prime number, starting with $p_0 = 2$.

We list some standard lemmata concerning the closure properties of the classes R_p and R_μ , which will be used in our analysis. Proofs of these results can be found in [10]–[12].

Lemma 1 (Standard Recursive Functions). The standard addition, multiplication and exponentiation over $\mathbb{N}$ are primitive recursive functions, as is the subtraction $\div : \mathbb{N}^2 \rightarrow \mathbb{N}$ over $\mathbb{N}$ defined by:

$$x \div y = \begin{cases} x - y, & \text{if } x \geq y \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

Lemma 2 (Definition by Cases). Let $R_1, R_2, \dots, R_m \subseteq \mathbb{N}^n$ be primitive recursive (μ -recursive) relations that partition $\mathbb{N}^n$ and $g_1, g_2, \dots, g_m : \mathbb{N}^n \rightarrow \mathbb{N}$ be primitive recursive (μ -recursive) functions. Then the function $f : \mathbb{N}^n \rightarrow \mathbb{N}$ defined by:

$$f(\bar{x}) = \begin{cases} g_1(\bar{x}), & \text{if } R_1(\bar{x}) \\ g_2(\bar{x}), & \text{if } R_2(\bar{x}) \\ \vdots \\ g_m(\bar{x}), & \text{if } R_m(\bar{x}) \end{cases} \quad (11)$$

is also primitive recursive (μ -recursive).

Lemma 3 (Bounded Minimization). *Let $R \subseteq \mathbb{N}^{n+1}$ be a relation. Define $D_{x,\bar{y}} = \{i \in \mathbb{N} \mid i \leq x \text{ and } R(i, \bar{y})\}$. The bounded minimization of R is defined as the function $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ with $f(x, \bar{y}) = \mu i \leq x : (R(i, \bar{y}))$ where:*

$$\mu i \leq x : (R(i, \bar{y})) = \begin{cases} \min D_{x,\bar{y}}, & \text{if } D_{x,\bar{y}} \neq \emptyset \\ x + 1, & \text{otherwise} \end{cases} \quad (12)$$

If the relation R is primitive recursive (μ -recursive), then the function $f(x, \bar{y}) = \mu i \leq x : (R(i, \bar{y}))$ is also primitive recursive (μ -recursive).

Lemma 4 (Standard Recursive Relations). *The relations $=, \leq, \geq, <, > \subseteq \mathbb{N}^2$ are primitive recursive.*

Lemma 5 (Closure for Logical Connectives). *If the relations $P, Q \subseteq \mathbb{N}^2$ are primitive recursive (μ -recursive), then so are the relations $P \wedge Q, P \vee Q, P \rightarrow Q$ and $\neg P$.*

Lemma 6 (Closure for Bounded Quantifiers). *Let $R \subseteq \mathbb{N}^{n+1}$ be a primitive recursive (μ -recursive) relation. Then the relations:*

$$Q(z, \bar{y}) = \forall x \leq z : (R(x, \bar{y})) \quad (13)$$

$$P(z, \bar{y}) = \exists x \leq z : (R(x, \bar{y})) \quad (14)$$

are also primitive recursive (μ -recursive).

Lemma 7 (Minimization of Relations). *Let $R \subseteq \mathbb{N}^{n+1}$ be a relation. Define $D_{x,\bar{y}} = \{i \in \mathbb{N} \mid i \geq x \text{ and } R(i, \bar{y})\}$. The minimization of R is defined as the function $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ with $f(x, \bar{y}) = \mu i \geq x : (R(i, \bar{y}))$ where:*

$$\mu i \geq x : (R(i, \bar{y})) = \begin{cases} \min D_{x,\bar{y}}, & \text{if } D_{x,\bar{y}} \neq \emptyset \\ \perp, & \text{otherwise} \end{cases} \quad (15)$$

If $x = 0$ we simply write $\mu i : (R(i, \bar{y}))$ instead of $\mu i \geq x : (R(i, \bar{y}))$. If the relation R is μ -recursive, then the function $f(x, \bar{y}) = \mu i \geq x : (R(i, \bar{y}))$ is also μ -recursive.

Lemma 8 (Concatenation). *Let $\langle \rangle : \mathbb{N}^* \rightarrow \mathbb{N}$ be a primitive recursive encoding. There exists a primitive recursive function $* : \mathbb{N}^2 \rightarrow \mathbb{N}$, called concatenation, such that for all $\bar{x}, \bar{y} \in \mathbb{N}^*$ we have:*

$$\langle \bar{x} \rangle * \langle \bar{y} \rangle = \langle \bar{x}, \bar{y} \rangle \quad (16)$$

B. General Recursion

We now present the notion of general recursive functions. Informally, a function is said to be general recursive if it can be computed by an algorithm or by a computational machine. The most widely used formal model of computation is the *Turing machine*, introduced by Alan Turing in his seminal work on the Entscheidungsproblem [6]. Accordingly, we define general recursive functions with respect to this model.

Given a Turing machine $M = (Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F)$, where Q is the set of states, Γ and Σ are the tape and the input alphabet respectively, $\sqcup \in \Gamma$ is the blank symbol, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\text{Left}, \text{Right}\}$ is the transition function, $q_0 \in Q$ is the starting state and $F \subseteq Q$ is the set of terminating states, we

consider the set K of total states as the set of pairs $C = (\sigma, q)$ where:

- 1) $\sigma : \mathbb{Z} \rightarrow \Gamma$ is a function that encodes the tape content, with $\sigma(0)$ representing the symbol under the machine's head
- 2) $q \in Q$ is the current state of the machine

We will use the notation σ_w , for $w = w_0 w_1 \dots w_{L-1} \in \Gamma^*$ to denote the function from $\sigma_w : \mathbb{Z} \rightarrow \Gamma$ defined by:

$$\sigma_w(n) = \begin{cases} w_n, & \text{if } 0 \leq n < L \\ \sqcup, & \text{otherwise} \end{cases} \quad (17)$$

That is, σ_w is the function that describes the content of the tape of M when the word w is written on it and the head of M is positioned over the first symbol of w .

The computation relation $\vdash_M^* \subseteq K \times K$ is defined by:

$$C_1 \vdash_M^* C_2 \Leftrightarrow \text{there exists a computation of } M \text{ that starts in the total state } C_1 \text{ and terminates in the total state } C_2$$

Using the Turing machine model, we can define the class of *general recursive partial functions*. Fix an input alphabet Σ_R , a tape alphabet Γ_R , and two injective functions:

$$\text{Input} : \mathbb{N}^* \rightarrow \Sigma_R^*, \quad \text{Output} : \Gamma_R^* \rightarrow \mathbb{N}$$

The function Input encodes inputs $\bar{x} \in \mathbb{N}^*$ as words over Σ_R and the function Output decodes words over Γ_R into natural numbers.

Definition 9 (General Recursive Function). *A partial function $f : \mathbb{N}^n \rightarrow \mathbb{N}$ is called general recursive iff there exists a Turing machine $M = (Q, \Gamma_R, \sqcup, \Sigma_R, \delta, q_0, F)$, with $q_0 = n$, such that for every input $\bar{x} \in \mathbb{N}^n$ the following conditions hold:*

$$\begin{aligned} \bar{x} \in \text{dom}(f) &\Rightarrow \\ (\sigma_{\text{Input}(\bar{x})}, q_0) &\vdash_M^* (\sigma_w, q_f) \text{ for some } q_f \in F \\ \text{and } w \in \Gamma_R^* &\text{ with } \text{Output}(w) = f(\bar{x}) \end{aligned} \quad (18)$$

$$\begin{aligned} \bar{x} \notin \text{dom}(f) &\Rightarrow \\ (\sigma_{\text{Input}(\bar{x})}, q_0) &\not\vdash_M^* (\sigma_w, q_f) \text{ for any pair } (q_f, w) \in F \times \Gamma_R^* \end{aligned} \quad (19)$$

If both of the above conditions hold, we say that the Turing machine M computes the partial function f . The class of all general recursive functions is denoted by $R(\mathbb{N})$.

The restriction $q_0 = n$ in the above definition ensures that a given Turing machine does not compute two distinct partial functions of different arities, with the exception of the *empty functions* $\epsilon_n : \mathbb{N}^n \rightarrow \mathbb{N}$ and $\epsilon_m : \mathbb{N}^m \rightarrow \mathbb{N}$, defined by $\text{dom}(\epsilon_n) = \text{dom}(\epsilon_m) = \emptyset$ for $n \neq m$.

Without this restriction, it would be possible to construct a single machine computing multiple functions of different arities. For instance, a “sum” machine could compute both the binary function $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ defined by $f(x, y) = x + y$ and the ternary function $g : \mathbb{N}^3 \rightarrow \mathbb{N}$ defined by $g(x, y, z) = x + y + z$.

C. Normal Form Theorem

A fundamental result in the theory of recursive functions is *Kleene's normal form theorem*. The first complete and formal statement, along with a detailed proof, is presented in his book [9].

Theorem 1 (Normal Form Theorem). *There exists a primitive recursive function $U : \mathbb{N} \rightarrow \mathbb{N}$, primitive recursive relations $T_n \subseteq \mathbb{N}^{n+2}$ for every $n \in \mathbb{N}$ and injective primitive recursive functions $S_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$ for all $n, m \in \mathbb{N}$, such that the following conditions hold:*

- 1) *A partial function $f : \mathbb{N}^n \rightarrow \mathbb{N}$ is general recursive iff there exists $e \in \mathbb{N}$ such that:*

$$f(\bar{x}) = U(\mu y : (T_n(y, e, \bar{x}))), \quad \forall \bar{x} \in \mathbb{N}^n \quad (20)$$

If the above condition holds then e is called a code of f .

- 2) *For every $e \in \mathbb{N}$, $\bar{z} \in \mathbb{N}^m$ and $\bar{x} \in \mathbb{N}^n$:*

$$\begin{aligned} U(\mu y : (T_{m+n}(y, e, \bar{z}, \bar{x}))) &= \\ &= U(\mu y : (T_n(y, S_n^m(e, \bar{z}), \bar{x}))) \end{aligned} \quad (21)$$

From Theorem 1 it follows that $R(\mathbb{N}) \subseteq R_\mu$, since each $f \in R(\mathbb{N})$ can be constructed by composing U with a minimization of T_n , where n is equal to the arity of f . The inverse inclusion $R_\mu \subseteq R(\mathbb{N})$ can also be easily proven. Therefore we have $R(\mathbb{N}) = R_\mu$. This allows us to use the term *recursive function/relation* to refer to both μ -recursive and general recursive partial functions/relations, as they are equivalent.

Theorem 1 allows us to define the *universal recursive functions*.

Definition 10 (Universal Recursive Function). *The universal recursive functions $\varphi^n : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ are defined for each $n \in \mathbb{N}$ as:*

$$\varphi^n(e, \bar{x}) = U(\mu y : (T_n(y, e, \bar{x}))) \quad (22)$$

For a given $n \in \mathbb{N}$ every recursive function $f : \mathbb{N}^n \rightarrow \mathbb{N}$ can be expressed as $f(\bar{x}) = \varphi^n(e, \bar{x})$, where e is a code of f .

The S_n^m functions can be used to construct, using only primitive recursion, codes of arbitrarily complex recursive functions. This is expressed in the following lemma, the proof of which heavily relies on the S_n^m functions:

Lemma 9 (Effectiveness of Composition, Primitive Recursion and Minimization). *For every $m, n \in \mathbb{N}$ there exist primitive recursive functions $\text{Com}_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$, $\text{Rec}_n : \mathbb{N}^2 \rightarrow \mathbb{N}$ and $\text{Min}_n : \mathbb{N} \rightarrow \mathbb{N}$ such that the following conditions hold:*

- 1) *If $e_1, e_2, \dots, e_m$ are codes of $f_1, f_2, \dots, f_m : \mathbb{N}^n \rightarrow \mathbb{N}$ and e_g is a code of $g : \mathbb{N}^m \rightarrow \mathbb{N}$ then $\text{Com}_n^m(e_g, e_1, e_2, \dots, e_m)$ is a code of the composition $g(f_1(\bar{x}), f_2(\bar{x}), \dots, f_m(\bar{x}))$.*
- 2) *If e_g is a code of $g : \mathbb{N}^n \rightarrow \mathbb{N}$ and e_h is a code of $h : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ then $\text{Rec}_n(e_g, e_h)$ is a code of the function $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$, which is defined by primitive recursion on h with base case g .*
- 3) *If e_g is a code of $g : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ then $\text{Min}_n(e_g)$ is a code of the minimization $\mu i \geq x : (g(i, \bar{y}) = 0)$.*

Lemma 9 enables the construction of primitive recursive functions that compute codes for arbitrarily complex recursive functions. Given a set of base recursive functions $f_1, f_2, \dots, f_k$ with corresponding codes $e_1, e_2, \dots, e_k$, and a function g defined by a known sequence of compositions, primitive recursions, and minimizations on these base functions, we can easily define a primitive recursive function $F : \mathbb{N}^k \rightarrow \mathbb{N}$ such that $F(e_1, e_2, \dots, e_k)$ is a code of g . The function F is constructed by appropriately composing the functions Com_n^m , Rec_n , and Min_n according to the sequence of operations that defines g .

D. Recursive Functions over the Rational Numbers

We extend the notions of primitive and general recursive functions to the set of rational numbers, enabling the application of results from recursion theory to model algorithms that operate on rational inputs. To this end, we fix a primitive recursive encoding $\diamond : \mathbb{N}^* \rightarrow \mathbb{N}$ for the remainder of this article.

We begin by defining an encoding of the rational numbers into the natural numbers. Using this encoding, any rational $q \in \mathbb{Q}$ can be identified with a corresponding code $x \in \mathbb{N}$. This identification, together with recursive functions on natural numbers, allows us to define recursive functions of the form $f : \mathbb{N} \rightarrow \mathbb{Q}$ and $g : \mathbb{Q}^n \rightarrow \mathbb{Q}$.

Definition 11 (Encoding of Rationals). *Let $q = (-1)^s \frac{N}{D}$ be a rational number, where $s, N, D \in \mathbb{N}$ and $D \neq 0$. We will call $\langle s, N, D \rangle \in \mathbb{N}$ a code of q .*

We introduce the primitive recursive functions $s_{\mathbb{Q}}, N_{\mathbb{Q}}, D_{\mathbb{Q}} : \mathbb{N} \rightarrow \mathbb{N}$, which satisfy:

$$q = (-1)^{s_{\mathbb{Q}}(u)} \cdot \frac{N_{\mathbb{Q}}(u)}{D_{\mathbb{Q}}(u)} \quad (23)$$

whenever u is a code of q , by:

$$s_{\mathbb{Q}}(u) = (u)_0 \quad (24)$$

$$N_{\mathbb{Q}}(u) = (u)_1 \quad (25)$$

$$D_{\mathbb{Q}}(u) = (u)_2 \quad (26)$$

We also define the primitive recursive relation $\text{isRat} \subseteq \mathbb{N}$ by:

$$\begin{aligned} \text{isRat}(u) &\Leftrightarrow u \text{ is a code of some rational } q \\ &\Leftrightarrow \text{seq}(u) \wedge \text{lh}(u) = 3 \wedge (u)_2 \neq 0 \end{aligned} \quad (27)$$

Finally, we define the function $\text{rat} : \text{isRat} \rightarrow \mathbb{Q}$, with $\text{rat}(u) = q$ whenever u is a code of q , by:

$$\text{rat}(u) = (-1)^{s_{\mathbb{Q}}(u)} \cdot \frac{N_{\mathbb{Q}}(u)}{D_{\mathbb{Q}}(u)}$$

Note that a rational number q does not admit a unique code. For example, both $\langle 0, 5, 2 \rangle$ and $\langle 2, 10, 4 \rangle$ are codes of $\frac{5}{2}$. Furthermore, the functions $s_{\mathbb{Q}}, N_{\mathbb{Q}}$, and $D_{\mathbb{Q}}$ are defined on all inputs, including those for which $\neg \text{isRat}(u)$. However, their values in such cases are not meaningful for the encoding.

Definition 11 allows us to represent functions and relations on rational numbers using recursive functions and relations on natural numbers. In particular, the four basic arithmetic operations, as well as the relations of equality, strict order, and

non-strict order over the rationals, are all primitive recursive, as formalized in Lemma 10.

Lemma 10. *There exist primitive recursive functions $+_{\mathbb{Q}}, -_{\mathbb{Q}}, \cdot_{\mathbb{Q}}, /_{\mathbb{Q}} : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for all $x, y \in \mathbb{N}$ with $\text{isRat}(x)$ and $\text{isRat}(y)$ the following conditions hold:*

$$\text{rat}(x +_{\mathbb{Q}} y) = \text{rat}(x) + \text{rat}(y) \quad (28)$$

$$\text{rat}(x -_{\mathbb{Q}} y) = \text{rat}(x) - \text{rat}(y) \quad (29)$$

$$\text{rat}(x \cdot_{\mathbb{Q}} y) = \text{rat}(x) \cdot \text{rat}(y) \quad (30)$$

$$\text{rat}(x /_{\mathbb{Q}} y) = \begin{cases} \text{rat}(x) / \text{rat}(y), & \text{if } \text{rat}(y) \neq 0 \\ 0, & \text{if } \text{rat}(y) = 0 \end{cases} \quad (31)$$

Furthermore, the relations $=_{\mathbb{Q}}, \leq_{\mathbb{Q}}, <_{\mathbb{Q}} \subseteq \mathbb{N}^2$ defined by:

$$x =_{\mathbb{Q}} y \Leftrightarrow \text{isRat}(x) \wedge \text{isRat}(y) \wedge \text{rat}(x) = \text{rat}(y) \quad (32)$$

$$x \leq_{\mathbb{Q}} y \Leftrightarrow \text{isRat}(x) \wedge \text{isRat}(y) \wedge \text{rat}(x) \leq \text{rat}(y) \quad (33)$$

$$x <_{\mathbb{Q}} y \Leftrightarrow \text{isRat}(x) \wedge \text{isRat}(y) \wedge \text{rat}(x) < \text{rat}(y) \quad (34)$$

are primitive recursive.

Proof. We will show the existence of $+_{\mathbb{Q}}$ and $/_{\mathbb{Q}}$. The existence of $-_{\mathbb{Q}}$ and $\cdot_{\mathbb{Q}}$ can be established in a similar fashion. Let $x, y \in \mathbb{N}$ with $\text{isRat}(x)$ and $\text{isRat}(y)$. We have:

$$\text{rat}(x) + \text{rat}(y) = \quad (35)$$

$$\frac{(-1)^{s_{\mathbb{Q}}(x)} N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) + (-1)^{s_{\mathbb{Q}}(y)} N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)}{D_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y)} \quad (36)$$

We consider three cases for expression (36):

- 1) if $s_{\mathbb{Q}}(x) \equiv s_{\mathbb{Q}}(y) \pmod{2}$ then $\text{rat}(x)$ and $\text{rat}(y)$ have the same sign, and expression (36) can be written as:

$$(-1)^{s_{\mathbb{Q}}(x)} \frac{N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) + N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)}{D_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y)} \quad (37)$$

- 2) if $s_{\mathbb{Q}}(x) \not\equiv s_{\mathbb{Q}}(y) \pmod{2}$ and $N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) \geq N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)$ then expression (36) can be written as:

$$(-1)^{s_{\mathbb{Q}}(x)} \frac{N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) \dot{-} N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)}{D_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y)} \quad (38)$$

- 3) if $s_{\mathbb{Q}}(x) \not\equiv s_{\mathbb{Q}}(y) \pmod{2}$ and $N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) < N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)$ then expression (36) can be written as:

$$(-1)^{s_{\mathbb{Q}}(y)} \frac{N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x) \dot{-} N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y)}{D_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y)} \quad (39)$$

The congruence modulo 2 can be expressed with the equivalences:

$$s_{\mathbb{Q}}(x) \equiv s_{\mathbb{Q}}(y) \pmod{2} \Leftrightarrow \text{even}(s_{\mathbb{Q}}(x) + s_{\mathbb{Q}}(y)) \quad (40)$$

$$s_{\mathbb{Q}}(x) \not\equiv s_{\mathbb{Q}}(y) \pmod{2} \Leftrightarrow \text{odd}(s_{\mathbb{Q}}(x) + s_{\mathbb{Q}}(y)) \quad (41)$$

where the primitive recursive relations $\text{even}, \text{odd} \subseteq \mathbb{N}$ are defined via their characteristic functions by:

$$\begin{cases} \chi_{\text{even}}(0) = 1 \\ \chi_{\text{even}}(n+1) = 1 \dot{-} \chi_{\text{even}}(n) \end{cases} \quad (42)$$

$$\chi_{\text{odd}}(n) = 1 \dot{-} \chi_{\text{even}}(n) \quad (43)$$

Using these, the three cases are described by the primitive recursive relations:

$$R_1(x, y) \Leftrightarrow \text{even}(s_{\mathbb{Q}}(x) + s_{\mathbb{Q}}(y)) \quad (44)$$

$$R_2(x, y) \Leftrightarrow \text{odd}(s_{\mathbb{Q}}(x) + s_{\mathbb{Q}}(y)) \wedge N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) \geq N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x) \quad (45)$$

$$R_3(x, y) \Leftrightarrow \text{odd}(s_{\mathbb{Q}}(x) + s_{\mathbb{Q}}(y)) \wedge N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) < N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x) \quad (46)$$

Therefore, the sign, numerator and denominator of $\text{rat}(x) + \text{rat}(y)$ are given by the primitive recursive functions:

$$s_+(x, y) = \begin{cases} s_{\mathbb{Q}}(x), & \text{if } R_1(x, y) \\ s_{\mathbb{Q}}(x), & \text{if } R_2(x, y) \\ s_{\mathbb{Q}}(y), & \text{if } R_3(x, y) \end{cases} \quad (47)$$

$$N_+(x, y) = \begin{cases} N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) + N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x), & \text{if } R_1(x, y) \\ N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) \dot{-} N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x), & \text{if } R_2(x, y) \\ N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x) \dot{-} N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y), & \text{if } R_3(x, y) \end{cases} \quad (48)$$

$$D_+(x, y) = D_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) \quad (49)$$

Finally, $+_{\mathbb{Q}}$ can be defined by:

$$x +_{\mathbb{Q}} y = \langle s_+(x, y), N_+(x, y), D_+(x, y) \rangle \quad (50)$$

As for the operation $/_{\mathbb{Q}}$, if $\text{rat}(y) \neq 0$, we have:

$$\text{rat}(x) / \text{rat}(y) = (-1)^{s_{\mathbb{Q}}(x) + s_{\mathbb{Q}}(y)} \frac{N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y)}{N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)} \quad (51)$$

We define the primitive recursive functions:

$$s_/(x, y) = s_{\mathbb{Q}}(x) + s_{\mathbb{Q}}(y) \quad (52)$$

$$N_/(x, y) = N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) \quad (53)$$

$$D_/(x, y) = N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x) \quad (54)$$

The function $/_{\mathbb{Q}}$ can then be defined by:

$$x /_{\mathbb{Q}} y = \begin{cases} \langle s_/(x, y), N_/(x, y), D_/(x, y) \rangle, & \text{if } N_{\mathbb{Q}}(y) \neq 0 \\ 0, & \text{otherwise} \end{cases} \quad (55)$$

For the relations $=_{\mathbb{Q}}, \leq_{\mathbb{Q}}, <_{\mathbb{Q}}$ we have:

$$\begin{aligned} \text{rat}(x) = \text{rat}(y) &\Leftrightarrow \\ &(\text{even}(s_{\mathbb{Q}}(x) + s_{\mathbb{Q}}(y)) \wedge N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) = N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)) \\ &\vee (N_{\mathbb{Q}}(x) = 0 \wedge N_{\mathbb{Q}}(y) = 0) \end{aligned} \quad (56)$$

$$\begin{aligned} \text{rat}(x) < \text{rat}(y) &\Leftrightarrow \\ &(\text{even}(s_{\mathbb{Q}}(x)) \wedge \text{even}(s_{\mathbb{Q}}(y)) \\ &\quad \wedge N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) < N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)) \\ &\vee (\text{odd}(s_{\mathbb{Q}}(x)) \wedge \text{odd}(s_{\mathbb{Q}}(y)) \\ &\quad \wedge N_{\mathbb{Q}}(x) D_{\mathbb{Q}}(y) > N_{\mathbb{Q}}(y) D_{\mathbb{Q}}(x)) \\ &\vee (\text{odd}(s_{\mathbb{Q}}(x)) \wedge \text{even}(s_{\mathbb{Q}}(y)) \\ &\quad \wedge \neg(N_{\mathbb{Q}}(x) = 0 \wedge N_{\mathbb{Q}}(y) = 0)) \end{aligned} \quad (57)$$

$$\text{rat}(x) \leq \text{rat}(y) \Leftrightarrow (\text{rat}(x) = \text{rat}(y)) \vee (\text{rat}(x) < \text{rat}(y)) \quad (58)$$

Therefore, the relations:

$$\{(x, y) \in \mathbb{N}^2 \mid \text{rat}(x) = \text{rat}(y)\} \quad (59)$$

$$\{(x, y) \in \mathbb{N}^2 \mid \text{rat}(x) \leq \text{rat}(y)\} \quad (60)$$

$$\{(x, y) \in \mathbb{N}^2 \mid \text{rat}(x) < \text{rat}(y)\} \quad (61)$$

are primitive recursive. By expressions (32), (33), (34) we conclude that the relations $=_{\mathbb{Q}}$, $\leq_{\mathbb{Q}}$, $<_{\mathbb{Q}}$ are primitive recursive. $\square$

Finding the maximum of two rational numbers is also primitive recursive.

Lemma 11. *There exists a primitive recursive function $\max_{\mathbb{Q}} : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for all $x, y \in \mathbb{N}$ with $\text{isRat}(x)$ and $\text{isRat}(y)$, the number $\max_{\mathbb{Q}}(x, y)$ is a code of the rational number $\max\{\text{rat}(x), \text{rat}(y)\}$.*

Proof. $\max_{\mathbb{Q}}(x, y)$ can be defined by:

$$\max_{\mathbb{Q}}(x, y) = \begin{cases} x, & \text{if } y \leq_{\mathbb{Q}} x \\ y, & \text{otherwise} \end{cases} \quad (62)$$

Since $\leq_{\mathbb{Q}}$ is primitive recursive, $\max_{\mathbb{Q}}$ is also primitive recursive. $\square$

IV. COMPUTABILITY FRAMEWORK

We now introduce the notion of *computable real numbers* and extend the constructions of Subsection III-D to this domain. The class of computable real numbers was first defined by Turing in the same work in which he introduced Turing machines [6]. Since then, computable analysis has developed into a rich field in theoretical computer science, and many equivalent definitions have been proposed. Here, we present one based on μ -recursive functions. For a detailed treatment, we refer the reader to [18].

Definition 12 (Computable Real Number). *A number $\alpha \in \mathbb{R}$ is called computable iff there exist recursive functions $s, N, D : \mathbb{N} \rightarrow \mathbb{N}$ with $D(n) \neq 0$ for all $n \in \mathbb{N}$ that satisfy:*

$$\left| \alpha - (-1)^{s(n)} \cdot \frac{N(n)}{D(n)} \right| < \frac{1}{2^n}, \quad \forall n \in \mathbb{N} \quad (63)$$

The set of all computable real numbers is denoted by $\mathbb{R}_c$.

Using Definition 12 and Theorem 1, we can define an encoding of the computable real numbers into the natural numbers, analogous to the encoding of Definition 11. This encoding then allows us to extend the constructions of Subsection III and to formalize recursion over computable real numbers.

Definition 13 (Encoding of Computables). *Let $\alpha \in \mathbb{R}_c$ with corresponding recursive functions $s, N, D : \mathbb{N} \rightarrow \mathbb{N}$ satisfying $D(n) \neq 0$ for all $n \in \mathbb{N}$ and:*

$$\left| \alpha - (-1)^{s(n)} \cdot \frac{N(n)}{D(n)} \right| < \frac{1}{2^n}, \quad \forall n \in \mathbb{N} \quad (64)$$

We call the recursive function $f : \mathbb{N} \rightarrow \mathbb{N}$ defined by:

$$f(n) = \langle s(n), N(n), D(n) \rangle \quad (65)$$

a recursive rational approximation of α and we refer to each code of f as a code of α .

We define the relation $\text{isCom} \subseteq \mathbb{N}$ by:

$$\text{isCom}(u) \Leftrightarrow u \text{ is a code of some computable real } \alpha \quad (66)$$

and the function $\text{com} : \text{isCom} \rightarrow \mathbb{R}_c$ that satisfies $\text{com}(u) = \alpha$ whenever u is a code of α by:

$$\text{com}(u) = \lim_{n \rightarrow \infty} \text{rat}(\varphi^1(u, n)) \quad (67)$$

We can now formulate the recursiveness of addition and multiplication over the computable real numbers, as stated in Lemma 12.

Lemma 12. *There exist primitive recursive functions $+_{\mathbb{R}_c}, \cdot_{\mathbb{R}_c} : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for all $x, y \in \mathbb{N}$ with $\text{isCom}(x)$ and $\text{isCom}(y)$ the following conditions hold:*

$$\text{com}(x +_{\mathbb{R}_c} y) = \text{com}(x) + \text{com}(y) \quad (68)$$

$$\text{com}(x \cdot_{\mathbb{R}_c} y) = \text{com}(x) \cdot \text{com}(y) \quad (69)$$

Proof. Let $x, y \in \mathbb{N}$ with $\text{isCom}(x)$ and $\text{isCom}(y)$. Set $\alpha = \text{com}(x) \in \mathbb{R}_c$ and $\beta = \text{com}(y) \in \mathbb{R}_c$. Denote by f_α, f_β the recursive rational approximations of α and β respectively:

$$f_\alpha(n) = \varphi^1(x, n) \quad (70)$$

$$f_\beta(n) = \varphi^1(y, n) \quad (71)$$

Let $\epsilon_\alpha(n) = \text{rat}(f_\alpha(n)) - \alpha$ and $\epsilon_\beta(n) = \text{rat}(f_\beta(n)) - \beta$ denote the approximation errors. By definition we have for all $n \in \mathbb{N}$:

$$|\epsilon_\alpha(n)|, |\epsilon_\beta(n)| < \frac{1}{2^n} \quad (72)$$

For the addition, note that the function $\text{rat}(f_\alpha(n+1)) + \text{rat}(f_\beta(n+1))$ satisfies:

$$|\text{rat}(f_\alpha(n+1)) + \text{rat}(f_\beta(n+1)) - (\alpha + \beta)| = \quad (73)$$

$$|\epsilon_\alpha(n+1) + \epsilon_\beta(n+1)| < \frac{1}{2^n} \quad (74)$$

Therefore, the function:

$$f_+(n) = f_\alpha(n+1) +_{\mathbb{Q}} f_\beta(n+1) \quad (75)$$

is a recursive rational approximation of $\alpha + \beta$. By Lemma 9 and equation (75), we can construct a primitive recursive function $+_{\mathbb{R}_c}$ with $x +_{\mathbb{R}_c} y$ being a code of the function f_+ . Therefore, $x +_{\mathbb{R}_c} y$ is a code of the computable real number $\alpha + \beta$.

For the multiplication we have:

$$\text{rat}(f_\alpha(k)) \cdot \text{rat}(f_\beta(k)) = \quad (76)$$

$$\alpha\beta + \alpha\epsilon_\beta(k) + \beta\epsilon_\alpha(k) + \epsilon_\alpha(k)\epsilon_\beta(k) \quad (77)$$

Therefore, we have:

$$|\text{rat}(f_\alpha(k)) \cdot \text{rat}(f_\beta(k)) - \alpha\beta| \quad (78)$$

$$\leq |\alpha\epsilon_\beta(k)| + |\beta\epsilon_\alpha(k)| + |\epsilon_\alpha(k)\epsilon_\beta(k)| \quad (79)$$

$$< \frac{|\alpha| + |\beta|}{k} + \frac{1}{2^{k+1}} \quad (80)$$

We know that $\text{rat}(f_\alpha(0)) - 1 < \alpha < \text{rat}(f_\alpha(0)) + 1$. Hence:

$$|\alpha| < \max\{|\text{rat}(f_\alpha(0)) - 1|, |\text{rat}(f_\alpha(0)) + 1|\} \quad (81)$$

$$\leq |\text{rat}(f_\alpha(0))| + 1 \quad (82)$$

$$\leq N_{\mathbb{Q}}(\varphi^1(x, 0)) + 1 \quad (83)$$

Define the recursive function:

$$M(x) = N_{\mathbb{Q}}(\varphi^1(x, 0)) + 1 \quad (84)$$

By construction, M satisfies $M(x) > |\alpha|$, $M(y) > |\beta|$ and $M(n) \neq 0$. By setting $k = K(n, x, y) = (M(x) + M(y))2^{n+1} > n$ in the inequality (80) we achieve:

$$|\text{rat}(f_\alpha(K(n, x, y))) \cdot \text{rat}(f_\beta(K(n, x, y))) - \alpha\beta| \quad (85)$$

$$< \frac{|\alpha| + |\beta|}{(|\alpha| + |\beta|)2^{n+1}} + \frac{1}{2^{K(n, x, y)+1}} < \frac{1}{2^n} \quad (86)$$

Therefore, the function:

$$f.(x, y, n) = f_\alpha(K(n, x, y)) \cdot_{\mathbb{Q}} f_\beta(K(n, x, y)) \quad (87)$$

is a recursive rational approximation of $\alpha \cdot \beta$, when viewed as a function of only n . By Lemma 9 we can construct a primitive recursive function F , with $F(x, y)$ being a code of $f.(x, y, n)$. The function $\cdot_{\mathbb{R}_c}$ can then be defined by:

$$x \cdot_{\mathbb{R}_c} y = S_1^2(F(x, y), x, y) \quad (88)$$

□

Subtraction and division over $\mathbb{R}_c$ can similarly be shown to be recursive, although we will not require these operations in our analysis. Notably, the relations of equality and ordering of computable real numbers are not recursive.

Lemma 13. *The relations $=_{\mathbb{R}_c}$, $<_{\mathbb{R}_c} \subseteq \mathbb{N}^2$ defined by:*

$$x =_{\mathbb{R}_c} y \Leftrightarrow \text{isCom}(x) \wedge \text{isCom}(y) \wedge \text{com}(x) = \text{com}(y) \quad (89)$$

$$x <_{\mathbb{R}_c} y \Leftrightarrow \text{isCom}(x) \wedge \text{isCom}(y) \wedge \text{com}(x) < \text{com}(y) \quad (90)$$

are not recursive.

Proof. We will reduce the halting problem to both $=_{\mathbb{R}_c}$ and $<_{\mathbb{R}_c}$. Since the halting problem is famously non recursive, we conclude that $=_{\mathbb{R}_c}$ and $<_{\mathbb{R}_c}$ are also non recursive.

The halting problem can be formulated, using the universal recursive functions, as determining for arbitrary $e, x \in \mathbb{N}$ whether $\varphi^1(e, x) \downarrow$ or not. The fact that it is not solvable algorithmically is expressed as the non recursiveness of the relation $H \subseteq \mathbb{N}^2$ defined by:

$$H(e, x) \Leftrightarrow \varphi^1(e, x) \downarrow \quad (91)$$

We proceed to defining the relation $H(e, x)$ in terms of $=_{\mathbb{R}_c}$ and $<_{\mathbb{R}_c}$. Let $e, x \in \mathbb{N}$ be arbitrary natural numbers. Define the function:

$$f(e, x, k) = \begin{cases} 1, & \text{if } \exists y \leq k : (T_1(y, e, x)) \\ 0, & \text{otherwise} \end{cases} \quad (92)$$

f is primitive recursive, since the relation T_1 is primitive recursive by Theorem 1. Furthermore, $H(e, x)$ is true iff

$f(e, x, k) = 1$ for some $k \in \mathbb{N}$. Define a second function $f_M : \mathbb{N}^3 \rightarrow \mathbb{N}$ by the primitive recursion:

$$\begin{cases} f_M(0, e, x) = f(e, x, 0) \cdot \langle 0, 1, 1 \rangle \\ f_M(n+1, e, x) = f_M(n, e, x) +_{\mathbb{Q}} \\ \quad f(e, x, n+1) \cdot \langle 0, 1, 2^{n+1} \rangle \end{cases} \quad (93)$$

The definition of f_M is such that the number $f_M(n, e, x)$ is a code of the rational number:

$$q_{n,e,x} = \sum_{k=0}^n f(e, x, k) \cdot \frac{1}{2^k} \quad (94)$$

Therefore, the function $f'_M(n) = f_M(n, e, x)$ is a recursive rational approximation of some computable real number β , which is zero iff $f(e, x, n) = 0$ for all $n \in \mathbb{N}$ and positive otherwise. This means that:

$$\begin{cases} 0 = \beta \Leftrightarrow \neg H(e, x) \\ 0 < \beta \Leftrightarrow H(e, x) \end{cases} \quad (95)$$

Let c_0 be a code of the computable real number 0. Let c_M be a code of the recursive function $\hat{f}(e, x, n) = f_M(n, e, x)$. A code of the function $f'_M(n) = \hat{f}(e, x, n)$, and therefore a code of β is given by $S_1^2(c_M, e, x)$. Hence, we arrive at the equivalences:

$$H(e, x) \Leftrightarrow \neg(c_0 =_{\mathbb{R}_c} S_1^2(c_M, e, x)) \quad (96)$$

$$H(e, x) \Leftrightarrow c_0 <_{\mathbb{R}_c} S_1^2(c_M, e, x) \quad (97)$$

From the above expressions we see that if either of the relations $=_{\mathbb{R}_c}$, $<_{\mathbb{R}_c}$ is computable, then H is also computable, which is false. Therefore, by contradiction, $=_{\mathbb{R}_c}$ and $<_{\mathbb{R}_c}$ are not computable. □

In fact, even the comparison of computable real numbers with rationals is not recursive, as stated in Lemma 14.

Lemma 14. *The relation $<_{\mathbb{R}_c, \mathbb{Q}} \subseteq \mathbb{N}^2$ defined by:*

$$x <_{\mathbb{R}_c, \mathbb{Q}} y \Leftrightarrow \text{com}(x) < \text{rat}(y) \quad (98)$$

is not recursive.

Proof. We use the exact same logic as with the proof of Lemma 13. Let c_0 be a code of the rational number 0. Let $f(e, x, k)$ be the same as in the proof of Lemma 13. Define f_M by the primitive recursion:

$$\begin{cases} f_M(0, e, x) = f(e, x, 0) \cdot \langle 1, 1, 1 \rangle \\ f_M(n+1, e, x) = f_M(n, e, x) +_{\mathbb{Q}} \\ \quad f(e, x, n+1) \cdot \langle 1, 1, 2^{n+1} \rangle \end{cases} \quad (99)$$

so that $f_M(n, e, x)$ is a code of the rational number:

$$q_{n,e,x} = \sum_{k=0}^n f(e, x, k) \cdot \frac{-1}{2^n} \quad (100)$$

Let c_M be a code of the computable function $\hat{f}(e, x, n) = f_M(n, e, x)$. Then we have the equivalence:

$$H(e, x) \Leftrightarrow S_1^2(c_M, e, x) <_{\mathbb{R}_c, \mathbb{Q}} c_0 \quad (101)$$

from which we conclude that $<_{\mathbb{R}_c, \mathbb{Q}}$ is not computable. □

Although the order relation on computable real numbers is not recursive, it is still possible to compute the maximum of two computable reals in a recursive way.

Lemma 15. *There exists a primitive recursive function $\max_{\mathbb{R}_c} : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for all $x, y \in \mathbb{N}$ with $\text{isCom}(x)$ and $\text{isCom}(y)$, the number $\max_{\mathbb{R}_c}(x, y)$ is a code of the computable real number $\max\{\text{com}(x), \text{com}(y)\}$.*

Proof. Let $x, y \in \mathbb{N}$ satisfy $\text{isCom}(x)$ and $\text{isCom}(y)$. Set $\alpha = \text{com}(x)$, $\beta = \text{com}(y)$, $f_\alpha(n) = \varphi^1(x, n)$ and $f_\beta(n) = \varphi^1(y, n)$. By definition, we have for all $n \in \mathbb{N}$:

$$\alpha - \frac{1}{2^n} < \text{rat}(f_\alpha(n)) < \alpha + \frac{1}{2^n} \quad (102)$$

$$\beta - \frac{1}{2^n} < \text{rat}(f_\beta(n)) < \beta + \frac{1}{2^n} \quad (103)$$

It follows that:

$$\max\left\{\alpha - \frac{1}{2^n}, \beta - \frac{1}{2^n}\right\} < \max\{\text{rat}(f_\alpha(n)), \text{rat}(f_\beta(n))\} \quad (104)$$

$$< \max\left\{\alpha + \frac{1}{2^n}, \beta + \frac{1}{2^n}\right\} \quad (105)$$

$$\Leftrightarrow \max\{\alpha, \beta\} - \frac{1}{2^n} < \text{rat}\left(\max_{\mathbb{Q}}\{f_\alpha(n), f_\beta(n)\}\right) \quad (106)$$

$$< \max\{\alpha, \beta\} + \frac{1}{2^n} \quad (107)$$

Therefore, the function:

$$f_{\max}(n) = \max_{\mathbb{Q}}(f_\alpha(n), f_\beta(n)) \quad (108)$$

is a recursive rational approximation of $\max\{\alpha, \beta\}$. The existence of $\max_{\mathbb{R}_c}$ follows from equation (108) and Lemma 9. $\square$

V. PROBLEM FORMULATION AND SOLUTION

We now proceed with a precise formulation of the problem. Our goal is to determine whether the task of constructing capacity-achieving codes is computationally solvable, in the sense of Question 3, which is a more formal version of Question 2:

Question 3. *Does there exist a Turing machine $M = (Q, \Gamma, \square, \Sigma, \delta, q_0, F)$ such that, when given as input a DMC $p_{Y|X}$, a rate $R < C(p_{Y|X})$ and an error tolerance $\epsilon > 0$, all appropriately encoded over the input alphabet Σ , it outputs a description over the tape alphabet Γ of a block code $\mathcal{C}$ with rate at least R and with maximum block error probability $\lambda_{\max} < \epsilon$?*

Note that the channel probabilities $p_{Y|X}(y | x)$, as well as the numbers R and ϵ are generally real numbers. However, there is no injective encoding of the set $\mathbb{R}$ over any finite alphabet Σ , since $|\mathbb{R}| = 2^{\aleph_0} > \aleph_0 = |\Sigma^*|$. The most general subset of $\mathbb{R}$ that can be encoded over Σ and for which we can perform computations using a Turing machine, is the set of computable real numbers $\mathbb{R}_c$. To this end, we will prove that there exists a Turing machine M that satisfies the conditions of Question 3, for any DMC with computable probabilities $p_{Y|X}(y | x)$ and for any rational values of R and ϵ .

To do this, we will construct a μ -recursive function $\text{FindCode} : \mathbb{N}^3 \rightarrow \mathbb{N}$ that takes as inputs appropriate encodings of $p_{Y|X}$, R and ϵ over the natural numbers, and outputs an encoding over $\mathbb{N}$ of a block code that satisfies the required conditions. Since, by Theorem 1, Turing machines and μ -recursive functions are computationally equivalent, we conclude that the problem can be solved by a Turing machine.

We will then extend the result to cover the case where ϵ is an arbitrary computable real number, rather than just a rational, and we will also discuss how to generalize the result to allow R to be any computable real number as well.

A. Formulation with Pseudocode

We will first describe an algorithm that solves the problem using pseudocode and then construct the function FindCode based on this pseudocode. A naive first formulation is given by Algorithm 1.

Algorithm 1 Naive approach

Input: $p_{Y|X}, R, \epsilon$
Output: $\mathcal{C}$

```

1:  $n \leftarrow 1$ 
2: while True do
3:   for  $\mathcal{C} \in \text{CODES}(p_{Y|X}, \lceil 2^{nR} \rceil, n)$  do
4:      $\lambda \leftarrow \lambda_{\max}(\mathcal{C}, p_{Y|X})$ 
5:     if  $\lambda < \epsilon$  then
6:       return  $\mathcal{C}$ 
7:     end if
8:   end for
9:    $n \leftarrow n + 1$ 
10: end while

```

Algorithm 1 can be summarized as follows:

- 1) Initialize $n \leftarrow 1$.
- 2) Generate the list $\text{CODES}(p_{Y|X}, \lceil 2^{nR} \rceil, n)$ consisting of all $(\lceil 2^{nR} \rceil, n)$ block codes for the channel $p_{Y|X}$. This is possible because, for given $m = \lceil 2^{nR} \rceil$ and n , the number of such block codes is finite and equal to $m^{|X|^n} \cdot |Y|^{m \cdot n}$. Note that the rate of an (m, n) block code is defined as $\frac{\log_2 m}{n}$, so $(\lceil 2^{nR} \rceil, n)$ block codes have by definition rate at least R .
- 3) For each code $\mathcal{C} \in \text{CODES}(p_{Y|X}, \lceil 2^{nR} \rceil, n)$, compute the maximum block error probability of $\mathcal{C}$ under $p_{Y|X}$, denoted by $\lambda = \lambda_{\max}(\mathcal{C}, p_{Y|X})$. This step is feasible, since the operations required to compute λ are recursive. After computing λ , check whether $\lambda < \epsilon$. If the condition holds, return the code $\mathcal{C}$. Otherwise, proceed to the next code.
- 4) If none of the codes $\mathcal{C} \in \text{CODES}(p_{Y|X}, \lceil 2^{nR} \rceil, n)$ satisfies the condition, increment n by one and repeat the process for the new codeword length.

This algorithm proceeds in an exhaustive search fashion by enumerating all possible block codes for increasing codeword lengths n , starting from $n = 1$, until a code satisfying the error constraint is found. When $R < C(p_{Y|X})$, Shannon's channel coding theorem guarantees the existence of such codes for all $n \geq n_0$, for some threshold n_0 . Therefore, the algorithm is

guaranteed to terminate with a valid block code $\mathcal{C}$ such that $\lambda_{\max}(\mathcal{C}, p_{Y|X}) < \epsilon$.

The issue with Algorithm 1 is that $\lambda_{\max}$ is, in general, a computable real number, and the truth of the expression $\lambda_{\max} < \epsilon$ cannot be recursively decided, as stated in Lemma 14. For this reason, we modify the algorithm based on the observations of Lemma 16.

Lemma 16. *The following are true:*

- 1) *There exists a recursive function $\text{BLB} : \mathbb{N} \rightarrow \mathbb{N}$ (acronym for Binary Lower Bound) such that, if $c_\epsilon \in \mathbb{N}$ with $\text{rat}(c_\epsilon) = \epsilon > 0$, $\epsilon \in \mathbb{Q}$ then $\text{BLB}(c_\epsilon) \downarrow$ and $b = \text{BLB}(c_\epsilon) \in \mathbb{N}$ satisfies $2^{-b} < \epsilon$.*
- 2) *For a DMC $p_{Y|X}$ with $C(p_{Y|X}) \neq 0$ and a positive rate $R < C(p_{Y|X})$, there exists a $(\lceil 2^{nR} \rceil, n)$ block code $\mathcal{C}$ with $\lambda_{\max}(\mathcal{C}, p_{Y|X}) < 2^{-b-2}$, for any $b \in \mathbb{N}$.*
- 3) *For $c_\lambda \in \mathbb{N}$ with $\text{com}(c_\lambda) = \lambda < 2^{-b-2}$, $\lambda \in \mathbb{R}_c$ we have $\text{rat}(\varphi^1(c_\lambda, b+2)) < 2^{-b-1}$.*
- 4) *For $c_\lambda \in \mathbb{N}$ with $\text{com}(c_\lambda) = \lambda \in \mathbb{R}_c$ and $\text{rat}(\varphi^1(c_\lambda, b+2)) < 2^{-b-1}$ we have $\lambda < 2^{-b}$.*

Proof. 1) Since $\epsilon > 0$, it holds that $2^{-n} < \epsilon$ for all natural numbers $n > -\log(\epsilon)$. Therefore, the minimization:

$$\text{BLB}(c_\epsilon) = \mu i : \langle 0, 1, 2^i \rangle \leq_{\mathbb{Q}} c_\epsilon \quad (109)$$

converges and it returns a number b with $2^{-b} < \epsilon$.

- 2) It follows immediately from the channel coding theorem.
- 3) We have:

$$\text{rat}(\varphi^1(c_\lambda, b+2)) < \lambda + \frac{1}{2^{b+2}} < \frac{1}{2^{b+1}} \quad (110)$$

- 4) We have:

$$\lambda < \text{rat}(\varphi^1(c_\lambda, b+2)) + \frac{1}{2^{b+2}} \quad (111)$$

$$< \frac{1}{2^{b+1}} + \frac{1}{2^{b+2}} < \frac{1}{2^b} \quad (112)$$

□

With Lemma 16 in mind we proceed with the following reasoning:

- 1) Calculate $b = \text{BLB}(c_\epsilon)$.
- 2) There exists a block code $\mathcal{C}$ such that $\lambda = \lambda_{\max}(\mathcal{C}, p_{Y|X}) < 2^{-b-2}$. For any such code, it holds that $\text{rat}(\varphi^1(c_\lambda, b+2)) < 2^{-b-1}$, where c_λ is any code of $\lambda \in \mathbb{R}_c$.
- 3) We search through all possible block codes for $p_{Y|X}$ and return the first one that satisfies $\text{rat}(\varphi^1(c_\lambda, b+2)) < 2^{-b-1}$. By the previous point, we will eventually find such a code.
- 4) The returned code $\mathcal{C}$ also satisfies $\lambda_{\max}(\mathcal{C}, p_{Y|X}) < 2^{-b} < \epsilon$.

The modified pseudocode is presented in Algorithm 2.

Note that the condition $\text{rat}(\varphi^1(c_\lambda, k+2)) < 2^{-k-1}$ is sufficient for ensuring that the code satisfies $\lambda_{\max}(\mathcal{C}, p_{Y|X}) < 2^{-b-1}$, but it is not necessary. This means that the code $\mathcal{C}$ that is eventually returned, although guaranteed to meet the error constraint, may not be the first code in the enumeration that actually satisfies $\lambda_{\max}(\mathcal{C}, p_{Y|X}) < 2^{-b-1}$.

Algorithm 2 Working approach

Input: $p_{Y|X}, R, \epsilon$
Output: $\mathcal{C}$

```

1:  $b \leftarrow \text{BLB}(\epsilon)$ 
2:  $n \leftarrow 1$ 
3: while True do
4:   for  $\mathcal{C} \in \text{CODES}(p_{Y|X}, \lceil 2^{nR} \rceil, n)$  do
5:      $c_\lambda \leftarrow$  a code of  $\lambda_{\max}(\mathcal{C}, p_{Y|X})$ 
6:     if  $\text{rat}(\varphi^1(c_\lambda, b+2)) < 2^{-b-1}$  then
7:       return  $\mathcal{C}$ 
8:     end if
9:   end for
10:   $n \leftarrow n + 1$ 
11: end while
```

In the following analysis, we will define the recursive function FindCode based on Algorithm 2. We will gradually construct intermediate recursive functions and relations that solve smaller parts of the problem. In the end, we will combine everything to achieve the full solution.

Specifically, we will proceed through the following steps:

- 1) Define an encoding of the class of all DMCs over the natural numbers.
- 2) Define an encoding of the class of all block codes over the natural numbers.
- 3) Construct a function Codes : $\mathbb{N}^4 \rightarrow \mathbb{N}$ that takes as input the sizes M and N of the input and output alphabets of a DMC $p_{Y|X}$, along with parameters m and n , and returns an encoding of the list of all (m, n) block codes for $p_{Y|X}$.
- 4) Construct a function $\Lambda : \mathbb{N}^2 \rightarrow \mathbb{N}$ that takes as input the encodings of a code $\mathcal{C}$ and a DMC $p_{Y|X}$ and outputs a code of the computable number $\lambda_{\max}(\mathcal{C}, p_{Y|X})$.
- 5) Construct a function AchievesError : $\mathbb{N}^3 \rightarrow \mathbb{N}$ that takes as input the code $c = \langle c_0, c_1, \dots, c_{k-1} \rangle$ where c_i are all encodings of block codes, an encoding c_H of a DMC and a number $b \in \mathbb{N}$. The function AchievesError returns the index i of the first code c_i that satisfies $\text{rat}(\varphi^1(\Lambda(c_i, c_H), b+2)) < 2^{-b-1}$, if such a code exists. Otherwise it returns k .
- 6) Construct a function MessageNumber : $\mathbb{N}^2 \rightarrow \mathbb{N}$, which satisfies $\text{MessageNumber}(c_R, n) = \lceil 2^{nR} \rceil$ for $R = \text{rat}(c_R) > 0$.
- 7) Combine the above to define the function FindCode.

B. Encoding of DMCs

In this subsection we define an encoding function $\text{Code}_{\mathcal{H}}$ from the set of all DMCs with computable probabilities to the set $\mathbb{N}$ of natural numbers. This is a crucial step, as we aim to represent Algorithm 2 by a recursive function, which means that we have to encode all the inputs of the algorithm into natural numbers.

We will identify a DMC with input alphabet X and output alphabet Y by its corresponding conditional distribution $p_{Y|X}$. We will further use the notation $[k] = \{1, 2, \dots, k\}$ for $k \in \mathbb{N} \setminus \{0\}$ to denote the set of the first k positive integers.

A DMC $p_{Y|X}$ with $X = \{x_1, x_2, \dots, x_M\}$ and $Y = \{y_1, y_2, \dots, y_N\}$ can be represented by a matrix $H \in \mathbb{R}^{M \times N}$, with $H_{ij} = p_{Y|X}(y_j | x_i)$. We denote by $\mathcal{H}$ the set of all matrices that represent DMCs with computable probabilities, as:

$$\mathcal{H} = \bigcup_{\substack{M, N \in \mathbb{N} \\ M, N \neq 0}} \left\{ H \in \mathbb{I}_c^{M \times N} \left| \sum_{j=1}^N H_{ij} = 1, \text{ for all } 1 \leq i \leq M \right. \right\} \quad (113)$$

where $\mathbb{I}_c = [0, 1] \cap \mathbb{R}_c$ denotes the set of computable real numbers in the interval $[0, 1]$.

We define an encoding $\text{Code}_{\mathcal{H}} : \mathcal{H} \rightarrow \mathbb{N}$. For $H \in \mathcal{H}$ with dimensions $M \times N$, choose some $x_{ij} \in \mathbb{N}$ with $\text{com}(x_{ij}) = H_{ij}$ for all elements H_{ij} of H . Then define:

- 1) $r(H, i) = \langle x_{i1}, x_{i2}, \dots, x_{iN} \rangle$ for all $i \in [M]$
- 2) $\text{Code}_{\mathcal{H}}(H) = \langle r(H, 1), r(H, 2), \dots, r(H, M) \rangle$

We also define the recursive functions $\text{RowNumber} : \mathbb{N} \rightarrow \mathbb{N}$ and $\text{Element} : \mathbb{N}^3 \rightarrow \mathbb{N}$ by:

$$\text{RowNumber}(c_H) = \text{lh}(c_H) \quad (114)$$

$$\text{ColumnNumber}(c_H) = \text{lh}((c_H)_0) \quad (115)$$

$$\text{Element}(c_H, i, j) = (c_H)_{i-1, j-1} \quad (116)$$

If $c_H = \text{Code}_{\mathcal{H}}(H)$ for some $H \in \mathcal{H}$ with dimensions $M \times N$, then $\text{RowNumber}(c_H) = M$, $\text{ColumnNumber}(c_H) = N$ and $\text{Element}(c_H, i, j)$ is a code of the computable real number H_{ij} , provided that $i \in [M]$ and $j \in [N]$.

C. Encoding of Block Codes

Building on the approach of Subsection V-B, we now define an encoding $\text{Code}_{\mathcal{C}}$ that maps the set of all block codes to the set $\mathbb{N}$ of natural numbers, thereby enabling the use of block codes as inputs to recursive functions. Since a rational error tolerance ϵ can also be encoded as a natural number, as discussed in Subsection III-D, this construction allows all inputs to Algorithm 2 to be represented using natural numbers.

For a (m, n) block code $\mathcal{C} = (E, D)$, where $E : \mathcal{M} \rightarrow X^n$ and $D : Y^n \rightarrow \mathcal{M}$ are the encoding and decoding functions, respectively, and $\mathcal{M}$ is the set of messages with $|\mathcal{M}| = m$, if $|X| = M$ and $|Y| = N$, then we can assume, without loss of generality, that $\mathcal{M} = [m]$, $X = [M]$ and $Y = [N]$.

We denote by $\mathbf{C}_{M, N, m, n}$ the set of all (m, n) block codes with input and output alphabets of sizes M and N respectively, as:

$$\mathbf{C}_{M, N, m, n} = \{(E, D) | E : [m] \rightarrow [M]^n, D : [N]^n \rightarrow [m]\} \quad (117)$$

We also denote by $\mathbf{C}$ the set of all block codes, as:

$$\mathbf{C} = \bigcup_{\substack{M, N, m, n \in \mathbb{N} \\ M, N, m, n \neq 0}} \mathbf{C}_{M, N, m, n} \quad (118)$$

We define an encoding $\text{Code}_{\mathbf{C}} : \mathbf{C} \rightarrow \mathbb{N}$. The definition is done in three steps:

- 1) We define Code_E , which encodes functions of the form $E : [m] \rightarrow [M]^n$. If $E(i) = (x_{i1}, x_{i2}, \dots, x_{in})$, define:

$$e_i = \langle x_{i1}, x_{i2}, \dots, x_{in} \rangle, \text{ for } i \in [m] \quad (119)$$

$$\text{Code}_E(E) = \langle e_1, e_2, \dots, e_m \rangle \quad (120)$$

- 2) Similarly, we define Code_D , which encodes functions of the form $D : [N]^n \rightarrow [m]$, as:

$$\text{Code}_D(D) = \langle D(\bar{y}_1), D(\bar{y}_2), \dots, D(\bar{y}_{N^n}) \rangle \quad (121)$$

where $\bar{y}_1, \bar{y}_2, \dots, \bar{y}_{N^n}$ is the lexicographic enumeration of all elements in $[N]^n$.

- 3) We define $\text{Code}_{\mathbf{C}}$ for all $\mathcal{C} = (E, D) \in \mathbf{C}$ as:

$$\text{Code}_{\mathbf{C}}(\mathcal{C}) = \langle \text{Code}_E(E), \text{Code}_D(D) \rangle \quad (122)$$

We will also need a recursive way to check whether a number $n \in \mathbb{N}$ is an encoding of some block code $\mathcal{C} \in \mathbf{C}$. For this reason, we define the primitive recursive relations:

- 1) $\text{isCode}_E \subseteq \mathbb{N}^5$, with $\text{isCode}_E(c, M, N, m, n)$ true iff $c = \text{Code}_E(E)$ for some function $E : [m] \rightarrow [M]^n$.

We have:

$$\text{isCode}_E(c, M, N, m, n) \Leftrightarrow$$

$$\begin{aligned} & M > 0 \wedge m > 0 \wedge n > 0 \wedge \text{seq}(c) \wedge \text{lh}(c) = m \\ & \wedge \forall i \leq m \div 1 : (\text{seq}((c)_i) \wedge \text{lh}((c)_i) = n \\ & \wedge \forall j \leq n \div 1 : ((c)_{i,j} \geq 1 \wedge (c)_{i,j} \leq M)) \end{aligned} \quad (123)$$

- 2) $\text{isCode}_D \subseteq \mathbb{N}^5$, with $\text{isCode}_D(c, M, N, m, n)$ true iff $c = \text{Code}_D(D)$ for some function $D : [N]^n \rightarrow [m]$.

We have:

$$\text{isCode}_D(c, M, N, m, n) \Leftrightarrow$$

$$\begin{aligned} & N > 0 \wedge m > 0 \wedge n > 0 \wedge \text{seq}(c) \wedge \text{lh}(c) = N^n \\ & \wedge \forall i \leq N^n \div 1 : ((c)_i \geq 1 \wedge (c)_i \leq m) \end{aligned} \quad (124)$$

- 3) $\text{isCode}_{\mathbf{C}} \subseteq \mathbb{N}^5$, with $\text{isCode}_{\mathbf{C}}(c, M, N, m, n)$ true iff $c = \text{Code}_{\mathbf{C}}(\mathcal{C})$ for some block code $\mathcal{C} \in \mathbf{C}_{M, N, m, n}$. We have:

$$\text{isCode}_{\mathbf{C}}(c, M, N, m, n) \Leftrightarrow$$

$$\begin{aligned} & \text{seq}(c) \wedge \text{lh}(c) = 2 \\ & \wedge \text{isCode}_E((c)_0, M, N, m, n) \\ & \wedge \text{isCode}_D((c)_1, M, N, m, n) \end{aligned} \quad (125)$$

Note that, since $\diamond$ is injective, the encoding $\text{isCode}_{\mathbf{C}}$ is a bijection from $\mathbf{C}$ to $\text{isCode}_{\mathbf{C}} \subseteq \mathbb{N}$.

D. Encoding of the Set of all (m, n) Block Codes

In this subsection we construct a recursive function $\text{Codes} : \mathbb{N}^4 \rightarrow \mathbb{N}$, which takes as input four natural numbers M, N, m, n and returns an encoding of the list of all (m, n) block codes for a DMC $p_{Y|X}$ with $|X| = M$ and $|Y| = n$. This function allows us to represent the set $\text{CODES}(p_{Y|X}, \lceil 2^{nR} \rceil, n)$ in line 4 of Algorithm 2 within the recursion framework.

Recall that for finite sets A and B , the number of functions $f : A \rightarrow B$ is $|B|^{|A|}$. Since a block code $\mathcal{C} \in \mathbf{C}_{M, N, m, n}$ can be constructed by pairing any two functions $E : [m] \rightarrow [M]^n$ and $D : [N]^n \rightarrow [m]$, we see that:

$$|\mathbf{C}_{M, N, m, n}| = (M^n)^m \cdot m^{N^n} = M^{mn} m^{N^n} \quad (126)$$

With this in mind, our goal is to define a primitive recursive function $\text{ParCodes} : \mathbb{N}^5 \rightarrow \mathbb{N}$, where $\text{ParCodes}(y, M, N, m, n)$ encodes the list of the first y block

codes in $\mathbf{C}_{M,N,m,n}$. This will be done inductively, using primitive recursion. Given $\text{ParCodes}(y, M, N, m, n)$, we construct $\text{ParCodes}(y+1, M, N, m, n)$ by appending the smallest number n that encodes a valid block code in $\mathbf{C}_{M,N,m,n}$ and has not yet appeared in the list.

We define the primitive recursive relation $\text{notInside} \subseteq \mathbb{N}^2$:

$$\begin{aligned} \text{notInside}(e, c) \Leftrightarrow \\ \text{seq}(c) \wedge (\text{lh}(c) > 0 \rightarrow \forall i \leq \text{lh}(c) \div 1 : ((c)_i \neq e)) \end{aligned} \quad (127)$$

$\text{notInside}(e, c)$ is true iff c is a code of a sequence that does not contain e . ParCodes can be defined by:

$$\text{ParCodes}(0, M, N, m, n) = \langle \varepsilon \rangle \quad (128)$$

$$\begin{aligned} \text{ParCodes}(y+1, M, N, m, n) = \\ \text{app}(\text{ParCodes}(y, M, N, m, n), u) \end{aligned} \quad (129)$$

where:

$$\begin{aligned} u = \mu i : (\text{isCode}_{\mathbf{C}}(i, M, N, m, n) \\ \wedge \text{notInside}(i, \text{ParCodes}(y, M, N, m, n))) \end{aligned} \quad (130)$$

Since $\text{Code}_{\mathbf{C}}$ is a bijection from $\mathbf{C}$ to $\text{isCode}_{\mathbf{C}}$, there are exactly $M^{mn}m^{Nn}$ numbers $i \in \mathbb{N}$ that satisfy $\text{isCode}(i, M, N, m, n)$. From this we can conclude that $\text{ParCodes}(y, M, N, m, n) \downarrow$ when $M, N, m, n \geq 1$ and $y \leq M^{mn}m^{Nn}$. Therefore, Codes can be defined by:

$$\text{Codes}(M, N, m, n) = \text{ParCodes}(M^{mn}m^{Nn}, M, N, m, n) \quad (131)$$

E. Calculating the Maximum Block Error Probability

In this subsection we define a recursive function $\Lambda : \mathbb{N}^2 \rightarrow \mathbb{N}$, which calculates the maximum block error probability of a block code $\mathcal{C}$ for some DMC $p_{Y|X}$. The inputs to Λ are two natural numbers that are the encodings of $\mathcal{C}$ and $p_{Y|X}$. The output is a code of the computable real number $\lambda_{\max}(\mathcal{C}, p_{Y|X})$. More precisely, we want to construct Λ so that it satisfies, for all $\mathcal{C} \in \mathbf{C}$ and $H \in \mathcal{H}$:

$$\text{com}(\Lambda(\text{Code}_{\mathbf{C}}(\mathcal{C}), \text{Code}_{\mathcal{H}}(H))) = \lambda_{\max}(\mathcal{C}, p_{Y|X}) \quad (132)$$

where $p_{Y|X}$ is the DMC represented by the matrix H .

To do so, we will first calculate the conditional distribution $p_{Y^n|X^n}$, which extends the DMC $p_{Y|X}$ to finite words of length n . Note that if $p_{Y|X}$ is represented by the matrix H , then $p_{Y^n|X^n}$ is represented by the n -th Kronecker power of H , denoted by $H^{\otimes n}$. This motivates the definition of $\text{Kron} : \mathbb{N}^2 \rightarrow \mathbb{N}$ which satisfies:

$$\text{Kron}(\text{Code}_{\mathcal{H}}(H), n) = \text{Code}_{\mathcal{H}}(H^{\otimes n}) \quad (133)$$

Suppose that $c_1 = \text{Code}_{\mathcal{H}}(H_1)$ and $c_2 = \text{Code}_{\mathcal{H}}(H_2)$ for some matrices $H_1, H_2 \in \mathcal{H}$, with dimensions $M_1 \times N_1$ and $M_2 \times N_2$ respectively. Denote by h_{1,i_1,j_1} and h_{2,i_2,j_2} the corresponding elements of the matrices H_1, H_2 . We define:

- 1) $h(c_1, c_2, i_1, i_2, j_1, j_2) = \text{Element}(c_1, i_1, j_1) \cdot \text{Element}(c_2, i_2, j_2)$, which computes a code of the computable number $h_{1,i_1,j_1} \cdot h_{2,i_2,j_2}$. This corresponds to the Kronecker product $[h_{1,i_1,j_1}] \otimes [h_{2,i_2,j_2}]$.

- 2) $\text{Row}_1(c_1, c_2, i_1, i_2, j_1)$, which computes an encoding of the Kronecker product $[h_{1,i_1,j_1}] \otimes [h_{2,i_2,1} \dots h_{2,i_2,N_2}]$. Row_1 is given by:

$$\begin{aligned} \text{Row}_1(c_1, c_2, i_1, i_2, j_1) = \\ R_1(\text{ColumnNumber}(c_2), c_1, c_2, i_1, i_2, j_1) \end{aligned} \quad (134)$$

where $R_1 : \mathbb{N}^6 \rightarrow \mathbb{N}$ is defined by primitive recursion as:

$$R_1(0, c_1, c_2, i_1, i_2, j_1) = \langle \varepsilon \rangle \quad (135)$$

$$\begin{aligned} R_1(y+1, c_1, c_2, i_1, i_2, j_1) = \\ \text{app}(R_1(y, c_1, c_2, i_1, i_2, j_1), \\ h(c_1, c_2, i_1, i_2, j_1, y+1)) \end{aligned} \quad (136)$$

- 3) $\text{Row}_2(c_1, c_2, i_1, i_2)$, which encodes the Kronecker product $[h_{1,i_1,1} \dots h_{1,i_1,N_1}] \otimes [h_{2,i_2,1} \dots h_{2,i_2,N_2}]$. It is defined by:

$$\begin{aligned} \text{Row}_2(c_1, c_2, i_1, i_2) = \\ R_2(\text{ColumnNumber}(c_1), c_1, c_2, i_1, i_2) \end{aligned} \quad (137)$$

where $R_2 : \mathbb{N}^5 \rightarrow \mathbb{N}$ is defined by primitive recursion as:

$$R_2(0, c_1, c_2, i_1, i_2) = \langle \varepsilon \rangle \quad (138)$$

$$\begin{aligned} R_2(y+1, c_1, c_2, i_1, i_2) = \\ R_2(y, c_1, c_2, i_1, i_2) * \text{Row}_1(c_1, c_2, i_1, i_2, y+1) \end{aligned} \quad (139)$$

- 4) $\text{Col}_1(c_1, c_2, i_1)$, which computes an encoding of the Kronecker product $[h_{1,i_1,1} \dots h_{1,i_1,N_1}] \otimes H_2$. Col_1 is given by:

$$\begin{aligned} \text{Col}_1(c_1, c_2, i_1) = \\ C_1(\text{RowNumber}(c_2), c_1, c_2, i_1) \end{aligned} \quad (140)$$

where $C_1 : \mathbb{N}^4 \rightarrow \mathbb{N}$ is defined by primitive recursion as:

$$C_1(0, c_1, c_2, i_1) = \langle \varepsilon \rangle \quad (141)$$

$$\begin{aligned} C_1(y+1, c_1, c_2, i_1) = \\ \text{app}(C_1(y, c_1, c_2, i_1), \text{Row}_2(c_1, c_2, i_1, y+1)) \end{aligned} \quad (142)$$

- 5) $\text{Col}_2(c_1, c_2)$, which computes an encoding of the Kronecker product $H_1 \otimes H_2$. Col_2 is given by:

$$\text{Col}_2(c_1, c_2) = C_2(\text{RowNumber}(c_1), c_1, c_2) \quad (143)$$

where $C_2 : \mathbb{N}^3 \rightarrow \mathbb{N}$ is defined by primitive recursion as:

$$C_2(0, c_1, c_2) = \langle \varepsilon \rangle \quad (144)$$

$$\begin{aligned} C_2(y+1, c_1, c_2) = \\ C_2(y, c_1, c_2) * \text{Col}_1(c_1, c_2, y+1) \end{aligned} \quad (145)$$

We can now define the primitive recursive function Kron as $\text{Kron}(c, n) = K(n \div 1, c)$, where $K : \mathbb{N}^2 \rightarrow \mathbb{N}$ is defined by:

$$\begin{aligned} \{ K(0, c) = c \\ K(y+1, c) = \text{Col}_2(K(y, c), c) \end{aligned} \quad (146)$$

Suppose that $c_C = \text{Code}_C(\mathcal{C})$ and $c_H = \text{Code}_H(H)$ for some block code $\mathcal{C} = (E, D) \in \mathbf{C}$ with message set $[m]$ and for some matrix $H \in \mathcal{H}$ representing a DMC $p_{Y|X}$. We define:

- 1) the primitive recursive function $\text{CodeLength} : \mathbb{N} \rightarrow \mathbb{N}$, which computes the codeword length of $\mathcal{C}$, by:

$$\text{CodeLength}(c_C) = \text{lh}((c_C)_{0,0}) \quad (147)$$

- 2) the primitive recursive function:

$$P(c_C, c_H, i, j) = \text{Element}(\text{Kron}(c_H, \text{CodeLength}(c_C)), i, j) \quad (148)$$

P calculates the probability of transition from the i -th input word to the j -th output word, when we use the block code $\mathcal{C}$ on the DMC $p_{Y|X}$.

- 3) the primitive recursive function $LO : \mathbb{N}^2 \rightarrow \mathbb{N}$, which calculates the order of a word $\bar{x} \in [M]^n$ in the lexicographic enumeration of the elements of $[M]^n$. In particular, it satisfies:

$$LO(\langle x_0, x_1, \dots, x_{n-1} \rangle, M) = 1 + \sum_{i=1}^n (x_{n-i} - 1) \cdot M^{i-1} \quad (149)$$

Note that, since all codewords of a given block code $\mathcal{C}$ have the same length n , it suffices to define LO so that it computes the lexicographical order of $\bar{x}$ over the set $[M]^n$, rather than over the commonly implied ordered set $[M]^{\leq n} = \bigcup_{0 \leq i \leq n} [M]^i$. Moreover, the length n can be directly derived from the code $\langle \bar{x} \rangle$, which means that the only required arguments for this function are $\langle \bar{x} \rangle$ and M . The function is defined as follows:

$$LO(c, M) = L(\text{lh}(c), c, M) \quad (150)$$

where the function $L : \mathbb{N}^3 \rightarrow \mathbb{N}$ is defined with primitive recursion by:

$$L(0, c, M) = 1 \quad (151)$$

$$L(y+1, c, M) = L(y, c, M) + ((c)_{\text{lh}(c) \div (y+1)} \div 1) \cdot M^y \quad (152)$$

- 4) the primitive recursive relation $W \subseteq \mathbb{N}^3$, by:

$$W(c_C, i, d) \Leftrightarrow (c_C)_{1, i \div 1} \neq d \quad (153)$$

$W(c_C, i, d)$ is true iff for the i -th word $\bar{y}$ in the lexicographic enumeration of $[N]^n$ we have $D(\bar{y}) \neq d$, where N is the size of the output alphabet Y .

- 5) the primitive recursive function $\Lambda_1 : \mathbb{N}^4 \rightarrow \mathbb{N}$ by:

$$\Lambda_1(0, c_C, c_H, d) = c_{0, \mathbb{R}_c} \quad (154)$$

$$\begin{aligned} \Lambda_1(i+1, c_C, c_H, d) &= \Lambda_1(i, c_C, c_H, d) + \mathbb{R}_c \\ &\quad \cdot W(c_C, i+1, d) \end{aligned} \quad (155)$$

where $c_{0, \mathbb{R}_c}$ is a code of the computable number 0. When $d \in [m]$, $\Lambda_1(i, c_C, c_H, d)$ calculates the first i terms of the sum:

$$\lambda(\mathcal{C}, p_{Y|X}, d) = \sum_{\bar{y} \in [N]^n : D(\bar{y}) \neq d} p_{Y^n|X^n}(\bar{y} | E(d)) \quad (156)$$

which represents the block error probability for a specific message $d \in \mathcal{M}$.

- 6) the primitive recursive function:

$$\begin{aligned} \Lambda_2(c_C, c_H, d) &= \\ \Lambda_1 \left(\text{ColumnNumber}(c_H)^{\text{CodeLength}(c_C)}, c_C, c_H, d \right) \end{aligned} \quad (157)$$

which calculates the whole sum of equation (156).

- 7) the primitive recursive function $\Lambda_3 : \mathbb{N}^3 \rightarrow \mathbb{N}$ by:

$$\Lambda_3(0, c_C, c_H) = c_{0, \mathbb{R}_c} \quad (158)$$

$$\begin{aligned} \Lambda_3(i+1, c_C, c_H) &= \\ \max_{\mathbb{R}_c} (\Lambda_3(i, c_C, c_H), \Lambda_2(c_C, c_H, i+1)) \end{aligned} \quad (159)$$

When $i \in [m]$ $\Lambda_3(i, c_C, c_H)$ calculates the maximum:

$$\max \{ \lambda(\mathcal{C}, p_{Y|X}, d) \mid 0 < d \leq i \} \quad (160)$$

The function Λ can finally be defined as:

$$\Lambda(c_C, c_H) = \Lambda_3(\text{lh}((c_C)_0), c_C, c_H) \quad (161)$$

F. Defining the Function AchievesError

In this subsection, we define a recursive function $\text{AchievesError} : \mathbb{N}^3 \rightarrow \mathbb{N}$ that determines whether there exists a block code in a given list $(\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_{k-1})$ which achieves a desired error tolerance $\epsilon = 2^{-b}$ for a given DMC $p_{Y|X}$. This function, together with $\text{Codes}(M, N, m, n)$, allows us to verify whether a specific codeword length n suffices to achieve the desired error 2^{-b} , or whether n must be increased according to Algorithm 2.

The inputs to AchievesError are the encoding $c = \langle c_0, c_1, \dots, c_{k-1} \rangle$, where $c_i = \text{Code}_C(\mathcal{C}_i)$, an encoding c_H of the DMC and the number b . Let $c_{\lambda, i} = \Lambda(c_i, c_H)$ be the code of the computable real $\lambda_{\max}(\mathcal{C}_i, p_{Y|X})$, as computed by the function Λ . If there exists a block code $\mathcal{C}_i$ in the list that achieves the condition $\text{rat}(\varphi^1(c_{\lambda, i}, b+2)) < 2^{-b-1}$ of Algorithm 2, then AchievesError returns the index i of the first such code. If no such code exists, then AchievesError returns k . More precisely, when $c = \langle c_0, c_1, \dots, c_{k-1} \rangle$, with $\text{isCode}_C(c_i)$, $\forall i$ we have:

$$\begin{aligned} \text{AchievesError}(c, c_H, b) &= \\ \begin{cases} \min \{ i < k \mid \text{rat}(\varphi^1(c_{\lambda, i}, b+2)) < 2^{-b-1} \}, & \text{if such } i \text{ exists} \\ k, & \text{otherwise} \end{cases} \end{aligned} \quad (162)$$

This function can be defined by:

$$\begin{aligned} \text{AchievesError}(c, c_H, b) &= \\ \mu i \leq \text{lh}(c) \div 1 \quad (\varphi^1(\Lambda((c)_i, c_H), b+2) <_{\mathbb{Q}} \langle 0, 1, 2^{b+1} \rangle) \end{aligned} \quad (163)$$

G. Defining the Function MessageNumber

We proceed to defining MessageNumber , with

$$\text{MessageNumber}(c_R, n) = \left\lceil 2^{n \cdot \text{rat}(c_R)} \right\rceil \quad (164)$$

whenever $\text{isRat}(c_R)$ and $\text{rat}(c_R) > 0$. Let $c_R \in \mathbb{N}$ be such a number. This function computes the cardinality of the message set $\mathcal{M}$ that a block code $\mathcal{C}$ with codeword length n must have in order to achieve a coding rate R . We have:

$$\left\lceil 2^{n \text{rat}(c_R)} \right\rceil = \min \left\{ i \in \mathbb{N} \mid 2^{n \frac{N_{\mathbb{Q}}(c_R)}{D_{\mathbb{Q}}(c_R)}} \leq i \right\} \quad (165)$$

$$= \min \left\{ i \in \mathbb{N} \mid 2^{n N_{\mathbb{Q}}(c_R)} \leq i^{D_{\mathbb{Q}}(c_R)} \right\} \quad (166)$$

Since $D_{\mathbb{Q}}(c_R) \geq 1$, the inequality $2^{n N_{\mathbb{Q}}(c_R)} \leq i^{D_{\mathbb{Q}}(c_R)}$ is true for $i = 2^{n N_{\mathbb{Q}}(c_R)}$. Therefore, equation (166) can be expressed with the bounded minimization:

$$\begin{aligned} \text{MessageNumber}(n, c_R) = \\ \mu i \leq 2^{n N_{\mathbb{Q}}(c_R)} : \left(2^{n N_{\mathbb{Q}}(c_R)} \leq i^{D_{\mathbb{Q}}(c_R)} \right) \end{aligned} \quad (167)$$

H. The recursive function FindCode

We will now combine the previously defined functions to construct the recursive function FindCode. When this function takes as input an encoding of a DMC $p_{Y|X}$ with $C(p_{Y|X}) \neq 0$, an encoding of a positive rate $R < C(p_{Y|X})$ and an encoding of a rational error tolerance $\epsilon > 0$, it returns an encoding of a $(\lceil 2^{nR} \rceil, n)$ block code $\mathcal{C}$ with $\lambda_{\max}(\mathcal{C}, p_{Y|X}) < \epsilon$.

We first define $C : \mathbb{N}^3 \rightarrow \mathbb{N}$ by:

$$\begin{aligned} C(c_H, c_R, n) = \\ \text{Codes}(\text{RowNumber}(c_H), \text{ColumnNumber}(c_H), \\ \text{MessageNumber}(n, c_R), n) \end{aligned} \quad (168)$$

$C(c_H, c_R, n)$ is the encoding of the sequence of all $(\lceil 2^{n \text{rat}(c_R)} \rceil, n)$ block codes for the DMC encoded by c_H .

We find the minimum required codeword length for achieving the target error probability ϵ by using a recursive function $\text{MinLength} : \mathbb{N}^3 \rightarrow \mathbb{N}$ defined by:

$$\begin{aligned} \text{MinLength}(c_H, c_R, c_\epsilon) = \\ \mu i : (\text{AchievesError}(C(c_H, c_R, i), c_H, \text{BLB}(c_\epsilon)) \\ < \text{lh}(C(c_H, c_R, i))) \end{aligned} \quad (169)$$

The function FindCode can finally be defined by:

$$\begin{aligned} \text{FindCode}(c_H, c_R, c_\epsilon) = \\ \text{AchievesError}(C(c_H, c_R, \text{MinLength}(c_H, c_R, c_\epsilon)), \\ c_H, \text{BLB}(c_\epsilon)) \end{aligned} \quad (170)$$

The function FindCode provides a general solution to the problem of finding capacity-achieving codes. Given as input encodings of a DMC $p_{Y|X}$, a target rate $R < C(p_{Y|X})$ and a rational error threshold $\epsilon > 0$, the function returns an encoding of a block code $\mathcal{C}$ for $p_{Y|X}$ that achieves a rate of at least R and satisfies $\lambda_{\max}(\mathcal{C}, p_{Y|X})$. Moreover, FindCode is recursive, as it is constructed by combining recursive functions using operations that preserve recursiveness.

I. Generalization for $\epsilon \in \mathbb{R}_c$

The function can be naturally extended to handle error thresholds $\epsilon \in \mathbb{R}_c$. This is accomplished by defining a recursive

function $\text{RLB} : \mathbb{N} \rightarrow \mathbb{N}$ (acronym for Rational Lower Bound), which, given a code c_ϵ of a computable real number $\epsilon > 0$, returns a code of a positive rational lower bound of ϵ . That is, $\text{isRat}(\text{RLB}(c_\epsilon))$ holds, and $\text{rat}(\text{RLB}(c_\epsilon)) = q \in \mathbb{Q}$ for some $0 < q < \epsilon$.

Lemma 17. *There exists a μ -recursive function $\text{RLB} : \mathbb{N} \rightarrow \mathbb{N}$ such that, if $\text{isCom}(c_\epsilon)$ and $\text{com}(c_\epsilon) = \epsilon > 0$, then:*

$$\text{RLB}(c_\epsilon) \downarrow \quad (171)$$

$$\text{isRat}(\text{RLB}(c_\epsilon)) \quad (172)$$

$$0 < \text{rat}(\text{RLB}(c_\epsilon)) < \epsilon \quad (173)$$

Proof. Let $c_\epsilon \in \mathbb{N}$ be as described above. Set $f_\epsilon(n) = \varphi^1(c_\epsilon, n)$. By definition 12 we have:

$$|\text{rat}(f_\epsilon(n)) - \epsilon| < \frac{1}{2^n}, \quad \forall n \in \mathbb{N} \quad (174)$$

This implies that for all $n \in \mathbb{N}$:

$$\epsilon - \frac{1}{2^{n-1}} < \text{rat}(f_\epsilon(n)) - \frac{1}{2^n} < \epsilon \quad (175)$$

Thus, $\text{rat}(f_\epsilon(n)) - \frac{1}{2^n}$ is always a rational lower bound of ϵ . In addition, when $n > \log(1/\epsilon) + 1$, the expression $\epsilon - \frac{1}{2^{n-1}}$ is strictly positive, and by inequality (175) we have:

$$\text{rat}(f_\epsilon(n)) - \frac{1}{2^n} > 0 \quad (176)$$

Therefore, there exist $n \in \mathbb{N}$ for which $\text{rat}(f_\epsilon(n)) - \frac{1}{2^n}$ is positive. Hence, the minimization:

$$L(c_\epsilon) = \mu n : (\varphi^1(c_\epsilon, n) -_{\mathbb{Q}} \langle 0, 1, 2^n \rangle >_{\mathbb{Q}} \langle 0, 0, 1 \rangle) \quad (177)$$

converges, and it returns a number $n = L(c_\epsilon)$ for which $\text{rat}(f_\epsilon(n)) - \frac{1}{2^n}$ is a positive rational lower bound of ϵ . Consequently, RLB can be defined by:

$$\text{RLB}(c_\epsilon) = \varphi^1(c_\epsilon, L(c_\epsilon)) -_{\mathbb{Q}} \langle 0, 1, 2^{L(c_\epsilon)} \rangle \quad (178)$$

□

Having the recursive function RLB, the extension of FindCode that works for $\epsilon \in \mathbb{R}_c$ is defined by:

$$\text{FindCodeExt}(c_H, c_R, c_\epsilon) = \text{FindCode}(c_H, c_R, \text{RLB}(c_\epsilon)) \quad (179)$$

If c_H is an encoding of a DMC $p_{Y|X}$, c_R is a code of a rational $R < C(p_{Y|X})$ and c_ϵ is a code of a computable real $\epsilon > 0$, then $\text{RLB}(c_\epsilon)$ is a code of a rational q with $0 < q < \epsilon$ and $\text{FindCodeExt}(c_H, c_R, c_\epsilon)$ is an encoding of a block code $\mathcal{C}$ for $p_{Y|X}$ with rate at least R and with maximum block error probability:

$$\lambda_{\max}(\mathcal{C}, p_{Y|X}) < q < \epsilon \quad (180)$$

J. Extension to $R \in \mathbb{R}_c$

A similar extension can be considered for the case where $R \in \mathbb{R}_c$. A natural attempt would be to define a function $\text{MessageNumberExt}(n, c_R) = \lceil 2^{nR} \rceil$, assuming c_R encodes a computable real number R . However, such a function cannot be defined recursively.

Lemma 18. *There does not exist any μ -recursive function $\text{MessageNumberExt} : \mathbb{N}^2 \rightarrow \mathbb{N}$ satisfying:*

$$\text{MessageNumberExt}(n, c_R) = \lceil 2^{nR} \rceil \quad (181)$$

for all $n, c_R \in \mathbb{N}$ and $R \in \mathbb{R}_c$ such that $\text{isCom}(c_R)$ and $\text{com}(c_R) = R > 0$.

Proof. By way of contradiction, suppose that there exists a recursive function MessageNumberExt with the above properties. Let c_M be defined as in the proof of Lemma 13 and let c_1 be a code of the computable real number 1. From expressions (96) and (97) we have:

$$\text{com}(c_1 +_{\mathbb{R}_c} S_1^2(c_M, e, x)) = 1 \Leftrightarrow \neg H(e, x) \quad (182)$$

$$\text{com}(c_1 +_{\mathbb{R}_c} S_1^2(c_M, e, x)) > 1 \Leftrightarrow H(e, x) \quad (183)$$

The above equivalences imply that:

$$H(e, x) \Leftrightarrow \left\lceil 2^{\text{com}(c_1 +_{\mathbb{R}_c} S_1^2(c_M, e, x))} \right\rceil = 2 \quad (184)$$

$$\Leftrightarrow \text{MessageNumberExt}(1, \text{com}(c_1 +_{\mathbb{R}_c} S_1^2(c_M, e, x))) = 2 \quad (185)$$

from which it follows that the relation H is recursive, which is a contradiction. Therefore, such a function MessageNumberExt does not exist. $\square$

This motivates an approach analogous to the one outlined in Subsection V-I, where the goal is to find a rational number $\hat{R}$ such that $R \leq \hat{R} < C(p_{Y|X})$, which can then be used as input to FindCodeExt . Achieving this requires the definition of two additional μ -recursive functions:

- 1) A function $\text{Capacity} : \mathbb{N} \rightarrow \mathbb{N}$, which takes as input an encoding of a DMC $p_{Y|X}$ and returns a code of its capacity $C(p_{Y|X})$. Computing the capacity of a DMC reduces to solving a convex optimization problem, which can be approached using methods such as the steepest descent algorithm. However, formally encoding such a procedure as a recursive function is beyond the scope of this paper.
- 2) A function $\text{RatInterpolation} : \mathbb{N}^2 \rightarrow \mathbb{N}$, which takes as input two codes c_α and c_β of computable real numbers α and β with $\alpha < \beta$, and returns a code of a rational number q satisfying $\alpha < q < \beta$. The construction of this function is provided in Lemma 19.

Lemma 19. *There exists a recursive function $\text{RatInterpolation} : \mathbb{N}^2 \rightarrow \mathbb{N}$ satisfying:*

$$\text{isRat}(\text{RatInterpolation}(c_\alpha, c_\beta)) \quad (186)$$

$$\text{com}(c_\alpha) < \text{RatInterpolation}(c_\alpha, c_\beta) < \text{com}(c_\beta) \quad (187)$$

for all $c_\alpha, c_\beta \in \mathbb{N}$ such that $\text{isCom}(c_\alpha)$, $\text{isCom}(c_\beta)$ and $\text{com}(c_\alpha) < \text{com}(c_\beta)$.

Proof. Let c_α, c_β be as above and set $\alpha = \text{com}(c_\alpha)$, $\beta = \text{com}(c_\beta)$. Define $f_\alpha(n) = \varphi^1(c_\alpha, n)$, $f_\beta(n) = \varphi^1(c_\beta, n)$ and $g(n) = (f_\alpha(n) +_{\mathbb{Q}} f_\beta(n)) /_{\mathbb{Q}} \langle 0, 2, 1 \rangle$. We will show that if:

$$\text{rat}(f_\beta(n)) - \text{rat}(f_\alpha(n)) > \frac{1}{2^{n-1}} \quad (188)$$

then the value $g(n)$ satisfies the desired inequality $\alpha < \text{rat}(g(n)) < \beta$. Suppose that the inequality (188) is true. Then, since $\text{rat}(g(n)) = \frac{\text{rat}(f_\alpha(n)) + \text{rat}(f_\beta(n))}{2}$, we have:

$$\alpha < \text{rat}(f_\alpha(n)) + \frac{1}{2^n} < \text{rat}(g(n)) < \text{rat}(f_\beta(n)) - \frac{1}{2^n} < \beta \quad (189)$$

Therefore, if the minimization:

$$\begin{aligned} N(c_\alpha, c_\beta) &= \mu n : (f_\alpha(n) -_{\mathbb{Q}} f_\beta(n) >_{\mathbb{Q}} \langle 0, 1, 2^{n-1} \rangle) \\ &= \mu n : (\varphi^1(c_\alpha, n) -_{\mathbb{Q}} \varphi^1(c_\beta, n) >_{\mathbb{Q}} \langle 0, 1, 2^{n-1} \rangle) \end{aligned} \quad (190)$$

converges, then the desired function RatInterpolation can be defined by:

$$\text{RatInterpolation}(c_\alpha, c_\beta) = g(N(c_\alpha, c_\beta)) \quad (192)$$

It now suffices to show that the minimization (191) converges. Set $\epsilon = \beta - \alpha > 0$. Note that for $n > \log(1/\epsilon) + 2 \Leftrightarrow \epsilon - \frac{1}{2^{n-1}} > \frac{1}{2^{n-1}}$ we have:

$$\text{rat}(f_\beta(n)) - \text{rat}(f_\alpha(n)) > \beta - \frac{1}{2^n} - \left(\alpha + \frac{1}{2^n} \right) \quad (193)$$

$$= \epsilon - \frac{1}{2^{n-1}} > \frac{1}{2^{n-1}} \quad (194)$$

From the above it follows that (191) converges. This concludes the proof. $\square$

VI. CONCLUSION

This work has established the existence of a Turing machine that solves the problem of constructing capacity-achieving codes. This machine takes as input a DMC $p_{Y|X}$, an error tolerance ϵ and a coding rate R , and in the case where $R < C(p_{Y|X})$, it outputs a block code $\mathcal{C}$ for $p_{Y|X}$ with rate at least R and $\lambda_{\max}(\mathcal{C}, p_{Y|X}) < \epsilon$. The construction works for the general case where all the transition probabilities of $p_{Y|X}$ and the tolerance ϵ are computable real numbers, and the rate R is a rational number. Furthermore, we discussed a generalization of this machine that works for $R \in \mathbb{R}_c$. These results demonstrate that there exist general algorithmic methods for constructing capacity-achieving codes that work for all DMCs.

While the proposed machines do solve the general problem, they rely on exhaustive search techniques and exhibit exponential complexity, rendering them impractical. Nevertheless, several refinements can be applied to slightly reduce the complexity of the resulting algorithms. First, the exponential overhead of the classical encoding $\diamond$ can be avoided either by employing alternative polynomial-time primitive recursive encodings, such as extensions of the Cantor pairing function discussed in [10], or by dispensing with such encodings altogether and instead implementing data structures that support efficient list operations. Second, the search space of block codes $\text{Codes}(M, N, m, n)$ can be significantly reduced by restricting attention to codes $\mathcal{C} = (E, D)$ in which the encoding function $E : \mathcal{M} \rightarrow X^n$ is injective. It is straightforward to show that any non-injective encoding function yields maximum error probability at least $\frac{1}{2}$; hence, since all

relevant cases satisfy $\epsilon < \frac{1}{2}$, no generality is lost. Furthermore, if $\mathcal{M} = \{s_1, s_2, \dots, s_m\}$, we may assume without loss of generality that the codewords $E(s_1), E(s_2), \dots, E(s_m)$ are ordered lexicographically, reducing the search space by an additional factor of $m!$. Finally, for many practical channels—including the binary symmetric channel and the binary erasure channel with crossover or erasure probability less than $\frac{1}{2}$ —the optimal decoding function $D : Y^n \rightarrow \mathcal{M}$ satisfies the minimum Hamming distance rule:

$$D(\bar{y}) \in \arg \min_{s \in \mathcal{M}} d_H(\bar{y}, E(s)), \quad (195)$$

where d_H denotes the Hamming distance. Thus, it suffices to restrict attention to codes consistent with this property. Nonetheless, even with all of these reductions, the search space remains exponential, and the resulting algorithms remain computationally infeasible.

The exponential nature of the algorithms does not diminish the significance of this work, since our goal is to establish the existence of a universal code-construction method, rather than to propose an efficient one. Having rigorously demonstrated the existence of such a method, future work may focus on identifying more efficient approaches or on characterizing possible trade-offs between algorithmic complexity and the generality of the channel models to which the method applies. For instance, although the consideration of computable real parameters is of theoretical interest, it is unnecessary in practice, since by the density of $\mathbb{Q}$ in $\mathbb{R}$, any channel can be approximated to arbitrary precision using a transition matrix with rational entries. Another natural direction is the algorithmic study of more general channel models, such as finite-state channels (FSCs).

APPENDIX A

DISCUSSION FOR CAPACITY-ACHIEVING SEQUENCES OF CODES

In this appendix, we compare the framework adopted in this work with that presented in [15]. The framework of [15] considers Turing machines that take as input a DMC and a parameter n , and output a block code of length n , whose error probability tends to zero and rate tends to the channel capacity as $n \rightarrow \infty$. In contrast, our machine takes as input a DMC together with explicit bounds on the code rate and error probability. Our approach, however, can be adapted to align with the framework of [15] by providing our machine with input parameters ϵ and R that asymptotically approach 0 and $C(p_{Y|X})$, respectively. Specifically, we describe the construction of a Turing machine that generates a sequence of block codes $\{\mathcal{C}_k\}$ for a given DMC $p_{Y|X}$, such that, as $k \rightarrow \infty$, the rate of $\mathcal{C}_k$ approaches the capacity $C(p_{Y|X})$ and its error probability tends to zero. This machine takes as input encodings of a DMC $p_{Y|X}$ with computable transition probabilities and a parameter $k \in \mathbb{N}$, and operates as follows:

- 1) It calculates the capacity of $p_{Y|X}$ using the function Capacity discussed in Subsection V-J.
- 2) It computes the computable real number $C(p_{Y|X}) - \frac{1}{k}$.
- 3) It employs the function RatInterpolation to determine a rational rate R satisfying $C(p_{Y|X}) - \frac{1}{k} < R < C(p_{Y|X})$.

- 4) It calls the function FindCode with inputs $p_{Y|X}$, the rate R , and an error tolerance $\epsilon_k = \frac{1}{k}$, and returns the resulting block code.

Since $R < C(p_{Y|X})$, the proposed machine halts and outputs a block code $\mathcal{C}_k$ with rate $R_k \geq R > C(p_{Y|X}) - \frac{1}{k}$ and maximum block error probability $\lambda_{\max}(\mathcal{C}_k, p_{Y|X}) < \frac{1}{k}$. It follows that the sequence $\{\mathcal{C}_k\}$ satisfies

$$\lim_{k \rightarrow \infty} R_k = C(p_{Y|X}) \quad (196)$$

$$\lim_{k \rightarrow \infty} \lambda_{\max}(\mathcal{C}_k, p_{Y|X}) = 0 \quad (197)$$

A key difference between the machine described above and the one proven impossible in [15] is that the latter requires the constructed code to have a block length exactly equal to n . In contrast, in our construction the parameter k is not directly tied to the codeword length. In fact, within the scope of this work, there are no evident bounds on the block length as a function of the parameter k . Furthermore, we employ a different formulation for describing input DMCs. In [15] the notion of a *computable family of channels* is used, which can represent a broader class of DMCs than those with merely computable real parameters. However, this formulation requires that the input space of DMCs have fixed input and output alphabet sizes M and N , and that the corresponding family of channels be expressible as a $M \times N$ matrix of *computable continuous* functions. In contrast, our formulation imposes no such restrictions, allowing arbitrary DMCs of any dimension to be provided as input to the same Turing machine.

REFERENCES

- [1] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [2] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 2003.
- [3] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
- [4] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2. IEEE, 1993, pp. 1064–1070.
- [5] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.
- [6] A. M. Turing *et al.*, "On computable numbers, with an application to the entscheidungsproblem," *J. of Math.*, vol. 58, no. 345–363, p. 5, 1936.
- [7] K. Gödel, "Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i," *Monatshefte für mathematik und physik*, vol. 38, no. 1, pp. 173–198, 1931.
- [8] S. C. Kleene, "General recursive functions of natural numbers," *Mathematische annalen*, vol. 112, no. 1, pp. 727–742, 1936.
- [9] —, "Introduction to metamathematics," 1952.
- [10] P. Odifreddi, *Classical recursion theory: The theory of functions and sets of natural numbers*. Elsevier, 1992, vol. 125.
- [11] H. Rogers Jr, *Theory of recursive functions and effective computability*. MIT press, 1987.
- [12] R. I. Soare, *Recursively enumerable sets and degrees: A study of computable functions and computably generated sets*. Springer Science & Business Media, 1999.
- [13] H. Boche and C. Deppe, "Computability of the zero-error capacity with kolmogorov oracle," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020.
- [14] M. Cheraghchi, "Capacity achieving codes from randomness conductors," in *2009 IEEE International Symposium on Information Theory*. IEEE, 2009, pp. 2639–2643.

- [15] H. Boche, R. F. Schaefer, and H. V. Poor, "Turing meets shannon: on the algorithmic construction of channel-aware codes," *IEEE Transactions on Communications*, vol. 70, no. 4, pp. 2256–2267, 2022.
- [16] S. C. Kleene, "Recursive predicates and quantifiers," *Transactions of the American mathematical Society*, vol. 53, no. 1, pp. 41–73, 1943.
- [17] —, "On notation for ordinal numbers," *The Journal of Symbolic Logic*, vol. 3, no. 4, pp. 150–155, 1938.
- [18] K. Weihrauch, *Computable analysis: an introduction*. Springer Science & Business Media, 2000.