Eötvös Loránd University

Faculty of Humanities

Doctoral Dissertation

Sóstai Zoltán

Empirical Constraints and the Computational Unpredictability of Physical Systems

Exploring the Physical Church–Turing thesis and the Halting Problem

DOI: 10.15476/ELTE.2024.176

Doctoral School of Philosophy	
Head of the Doctoral School:	Dsc, Prof. Tamás Ullmann
Doctoral Programme of Logic and Philosophy of Science	
Head of the Doctoral Programme:	Dsc, Prof. László E. Szabó
Members of the Assessment Committee:	
Chair:	PhD, Prof. Zsófia Zvolenszky, University Professor
Opponents:	PhD, Mihály Héder, Habilitated Associate Professor
	PhD, Márton Gömöri, Assistant Professor
Secretary:	PhD, Dániel Kodaj, Assistant Professor
Member:	PhD, Gergely Székely, Associate Professor
Subsidiary members:	PhD, Judit Madarász, Senior Research Fellow
	PhD, András Máté, Retired Associate Professor
Supervisor:	Dsc, Prof. László E. Szabó

Budapest, 2024

Table of contents

1. Introduction and problem description

The question of whether the physical world is computably predictable lies at the heart of the intersection between computational theory and the philosophy of science. The Church–Turing thesis (Copeland 2020), which states that any effectively calculable function can be computed by a Turing machine, has had a profound impact on our understanding of the nature of computation. When extended to the physical world through the physical Church–Turing thesis – or in short, the PCTT – (Deutsch 1985), which asserts that every finitely realizable physical system can be simulated by a Turing machine, it raises fundamental questions about the limits of what can be known and predicted about the behavior of physical systems.

Central to this inquiry is the halting problem (Turing 1936), a seminal result in computability theory that demonstrates the existence of undecidable problems – problems for which no algorithm can determine whether a given program will halt on a given input. According to a specific argument (Lloyd 2012) which we can call Argument A, when the halting problem is considered in conjunction with a variant of the physical Church–Turing thesis, it suggests that there may be physical processes whose outcomes cannot be predicted by any computational means.

This thesis delves into the profound implications of this argument, focusing on the core problem that arises when a variant of the physical Church–Turing thesis and the halting problem are taken to hold simultaneously. This assumption frames physical systems as computational systems capable of executing decision making processes. With the assumption of physicalism that only physical entities exist, it can be ensured that any physical decision process can be simulated on a Turing machine. Decisions are then defined as the outcomes of computational processes that resolve specific questions about states of affairs in the physical world. In physical systems viewed through the lens of the physical Church–Turing thesis, decision-making is equated to executing a computational algorithm that processes inputs to arrive at a conclusion.

Predicting the outcome of a computational process, especially in physical systems, involves computing that process's future state based on its initial conditions and its program. Predictions are therefore higher-order computational tasks that determine the results of decision-making processes. If every physical system can be simulated by a Turing machine, and if there are problems that are undecidable for Turing machines, then it seems to follow that there may be physical states or processes that are fundamentally unpredictable. In particular, some predictions – which are higher-order decisions regarding computational decisions – can't be carried out.

According to Argument A, because of the physical Church–Turing thesis, all limitations characteristic of Turing machines will also apply to physical systems. In this way, if a certain function, namely the halting function, cannot be calculated for Turing machines, then there cannot exist a physical computer system that can calculate it. Therefore, making predictions about computational processes directly invokes the halting problem, particularly when predictions aim to determine whether a decision process will halt or continue indefinitely. This unpredictability poses a challenge to the notion of computability and predictability in the physical world, as it suggests that there may be inherent limitations to what can be known or computed about certain physical systems. There may be physical processes that cannot be predicted at all.

Some might argue that unpredictability is not a significant concern because quantum mechanics already involves unpredictable outcomes. In quantum mechanics, unpredictability is rooted in the theory's probabilistic framework. Quantum states do not have definite properties before they are measured; instead, they are described by probabilities. The exact position or momentum of a particle is fundamentally uncertain until an observation is made which determines a definite outcome. This

type of unpredictability is about the intrinsic randomness of quantum events, which means that even with complete information about a quantum system, only probabilistic predictions are possible. The unpredictability from the halting problem is of a different nature. It arises not from physical randomness but from the logical and computational limits of decision-making processes. The halting problem demonstrates that there are some questions about computational processes that are undecidable — no algorithm can solve these problems for all possible cases. This means that for some computational tasks, it is not just that predictions are uncertain or probabilistic; it is that no prediction or conclusion can definitively be reached at all using any algorithm.

The consequences of this problem and Argument A can be far-reaching, extending beyond the realm of computational theory and into the foundations of scientific inquiry itself. If there are physical states that cannot be predicted, even in principle, then it raises questions about the nature of scientific explanation, the role of prediction in scientific theories, and the limits of empirical knowledge.

Moreover, if certain physical computations cannot be predicted, then it follows that their outcomes cannot be evaluated for correctness or meaningfulness. This poses a challenge to the notion of scientific verification and raises questions about the foundations of empirical knowledge. If there are physical processes whose outcomes are inherently unpredictable, then what kind of physical processes are these and how can we assess the validity of scientific theories that purport to describe and explain those processes?

This thesis aims to explore these questions in depth, drawing on the tools of computational theory, philosophy of science, and epistemology to shed light on the nature and implications of this core problem. By carefully examining the assumptions and consequences of the physical Church–Turing thesis and the halting problem, this work seeks to contribute to a deeper understanding of the limits of computability and predictability in the physical world, and to stimulate further research and dialogue on these fundamental questions.

The negative consequences that follow from Argument A primarily revolve around the epistemic limitations imposed on any physical system arising from computational decision-making processes. These consequences challenge several foundational philosophical and scientific assumptions about determinism, predictability, and the nature of knowledge. Due to the uncomputability of the halting problem applied to physical systems, some future decisions about physical states of affairs cannot be predicted. This unpredictability implies a fundamental limit to our ability to forecast the outcomes of certain physical and computational processes, even when they are deterministic in nature.

If certain decisions or outcomes cannot be predicted due to their computational complexity or self-referential nature, it follows that evaluating their correctness or meaningfulness is not possible. This is particularly problematic for computational models of the physical world, which rely on the ability to predict and verify outcomes to validate their usefulness.

The argument indirectly challenges the principles of empiricism, which holds that knowledge of the world can only be obtained through sensory experience and experimentation. If there exist physical facts (such as the halting behavior of a physical computation) that cannot be predicted or evaluated for correctness by any physical agent, this suggests a boundary to what can be known through empirical methods. Argument A may lead to a form of agnosticism regarding our ability to obtain certain kinds of knowledge about the physical world. This agnosticism pertains not just to specific future events or states of affairs but extends to a broader skepticism about our capacity to fully understand or predict complex systems. Additionally, if the argument necessitates considering non-physical explanations or entities to resolve its contradictions (as the document explores), this could pose a challenge to physicalism — the view that everything is physical or ascribable to physical processes.

The positive consequences that follow from Argument A include the philosophical implications of epistemic limitations for any physical system. By demonstrating that certain future-oriented computations are beyond predictive capabilities, it can provide explanations and a computational foundations for philosophical concepts like free will and qualia (the subjective quality of experiences). Argument A suggests that the unpredictable nature of decision-making in deterministic physical systems could underpin the experience of free will or the emergence of qualia, providing a bridge between physical determinism and subjective experience. Is Argument A valid? Does it really follow that physical agents can't predict some of their decisions?

The key thesis of this doctoral dissertation is to show that the core argument under investigation, which asserts that there are physical decisions whose outcome can't be predicted — Argument A — is false.

Following the presentation of Key Concepts in Chapter 2 used in arguments, Chapter 3 deals with the analysis of Argument A, where a counter-argument (Sóstai 2023) — called Argument B — and also an extension to this counter-argument are presented against it.

In order to check historically whether arguments similar to Argument A have been put forward and to check whether arguments similar to Argument B have been proposed against them, we find a noteworthy parallel in László Kalmár's critique of Church's Thesis in Chapter 4. Kalmár's enquiry, called '*Argument Against the Plausibility of Church's Thesis*' (Kalmár 1959) presents a philosophical challenge to the foundational assumptions of computational theory, arguing against a form of agnosticism itself. Kalmár argued against the completeness of Church's Thesis by proposing the existence of effectively calculable functions that are not encompassed by general recursive functions or Turing-computable functions. This contention mirrors the spirit of Argument A by questioning the limits of what can be predicted or computed within our current theoretical frameworks.

Kalmár's argument is based on the non-recursive function $\psi(x)$ which was found to be non-recursive by Kleene (Kleene 1936) and which is therefore not effectively calculable according to Church's Thesis. The $\psi(x)$ function itself is based on a recursive function $\phi(x,y) = 0$. Kalmár argued that because the function ϕ can be an elementary function, then it expresses a basic truth about physical reality. This was crucial for him to assert that his arguments hold a significant connection to 'objective reality.' He explained that *„every proposition (such as $\phi(x,y) = 0$) that contains only non negative integers and the basic arithmetic operations (addition, subtraction, multiplication and division) refers to quantitative relations in objective reality."* (Kalmár 1957a, 34) Kalmár thought that the function ψ, which is based on ϕ and which requires a solution for $\exists (y)(\phi(x,y) = 0)$, also expresses a truth or a property about physical reality. The problem with Church's Thesis arises if there exist absolutely undecidable propositions. $\exists (y)(\phi(x,y) = 0)$ can be considered as such, because it can not be proved, but the negation of it cannot be proved either. In Kalmár's words, it is absolutely undecidable for the following reason: *„As a matter of fact, the problem, if the proposition in question holds or not, does not contain any parameter and, supposing Church's Thesis, the proposition itself can be neither proved nor disproved, not only in the frame of a fixed postulate system, but even admitting any correct means."* (Kalmár 1959, 75) Such an absolutely undecidable proposition would lead to a world view where a basic truth about physical reality could not be obtained, which would contradict physicalism and empiricism and which would lead to a form of agnosticism about what can be known in physical reality.

The reception to Kalmár's critique, particularly from Mendelson, introduced an essential perspective on evaluating such agnostic arguments. Mendelson's critique focused on the implicit assumption of the effective enumerability of correct proofs, challenging the foundation of Kalmár's argument.

Kalmár's argument combined with Hypothesis H along Mendelson's critique (Kalmár 1959, 74) (Mendelson 1963, 203-204) can be summarized as follows: We assume Church's Thesis (or in other words, the Church–Turing thesis), according to which the effectively calculable functions are the recursive functions. We assume that the set of correct proofs are recursively enumerable. We also assume – following Kleene's proof or following Turing's proof – that there exists a function ψ which is non-recursive, which is therefore not effectively calculable or computable. Then this leads to a contradiction, because this implies that on the one hand we can provably compute that a statement such as $\exists (y)(\phi(x,y) = 0)$ is not effectively computable and on the other hand that there is no computable proof that such statements are not effectively calculable. This argument – we can call it Argument K – shows a strong similarity to the counter-argument to Argument A – Argument B –, particularly because Argument B applies the assumption of physicalism and the physical Church–Turing thesis, which imply that the set of physically correct proofs must be computably enumerable by a physical computer, which would then imply a similar contradiction for Turing's anti-diagonal proof as it implies for Kleene's proof of ψ being non-recursive.

Argument B in itself doesn't offer a solution. It only shows that there is something wrong with Argument A. This suggests that it might be the naive application of the PCTT, which disregards what is physically meaningful, which disregards any physical constraints on measurement and which applies an unbounded prediction which can be the source of contradiction of Argument A with physicalism and empiricism. Based on this suggestion, Chapter 5 analyzes the meaningfulness of physical computations, particularly focusing on the empirical constraints that such physical meaningfulness requires.

In Chapter 5, based on the definition of representation for physical formal systems (E. Szabó 2017, 8-9), two critical conditions (A and B) are used to establish a systematic physical and empirical basis for computations to meaningfully represent or predict physical states of affairs. Condition A requires a one-to-one correspondence between computational processes and physical realities, while Condition B demands that these computations accurately reflect the occurrence or non-occurrence of these realities through verifiable outcomes. Meaningful computations must halt with correct outputs and in order for a computation to be meaningful its halting behavior must be empirically verifiable – otherwise it would not be possible to decide if the computation in question corresponds to reality or not. This requirement aligns with the principles of empiricism, which emphasize observable and verifiable phenomena as the basis for knowledge. By accepting physicalism, it follows that any such evaluation of meaning must be carried out by a physical process. If according to a variant of the PCTT, every physical process must be computable and no computable process can determine the halting behavior of H, it follows that no physical process can effectively determine this either. This presents a contradiction, where physical processes, which are supposed to be capable of all computable actions, cannot evaluate something as fundamental as halting behavior. This is a significant problem, because neither D and H are meaningful from a common sense physicalist view and that evaluating halting behavior would be crucial for evaluating meaningfulness. A significant part of Chapter 5 deals with how this problem can be resolved.

The contradiction in Argument A arises from the problem of predictability in deterministic physical systems and the limits imposed by the halting problem. If a computational prediction is unbounded, then it operates with endless resources and doesn't pose any measurable condition for halting and non-halting. A solution to this problem can be given through the notion of bounded predictions, where the halting condition for these predictions is operationally defined. Operational definitions specify the exact procedures or actions required to measure a concept. For bounded predictions, operational definitions provide clear observable, measurable criteria for what constitutes a computation halting, a computational prediction being correct, or a physical state being accurately represented by a

computational process. This introduces an empirically measurable criterion for halting, whether or not a computation reaches a conclusion within these finite bounds.

A *t-finite* computational process is a computation that is bounded by a finite number of steps and states, denoted by t. A computation or a prediction that is about the halting behavior of another computation within the same *t-finite* class is inherently unpredictable. This is due to the constraints imposed by a finite variant of the diagonal argument of Turing's proof, where a computation cannot predict its own halting behavior or that of another computation within the same bounded conditions without running into paradoxes or contradictions.

However, there is a critical distinction when it comes to predictions made by a *t+1-finite* physical system about a *t-finite* system. A computational process that operates within a slightly larger bound ($t+1$) — meaning that it has access to more computational steps or states than the *t-finite* processes it is evaluating — can predict the halting behavior of those *t-finite* processes without contradiction. This distinction resolves the problem of agnosticism, i.e. the general predictability dilemma by suggesting that while computations are bound by certain limitations, they are not universally unpredictable. By requiring a measurable empirical criteria of halting, the problem of unpredictable physical states of affairs disappears. This has the consequence that all computations under empirical criteria of halting can be evaluated for correctness and meaningfulness.

Chapter 6 provides a comprehensive overview of philosophical arguments related to free will, particularly focusing on unpredictability and determinism. Karl Popper (Popper 1950, 1982) argued against scientific determinism, suggesting that it's impossible for any physical system, including the universe, to predict its own future states with complete accuracy. He illustrated this through the Tristram Shandy paradox, Gödelian arguments, and the Oedipus argument, each showing limitations in the ability of systems to predict their own states due to inherent delays in processing information about changes in state. D. M. MacKay (MacKay 1967) questioned the mechanistic view that human actions are inevitable and argued that free will is not an illusion. He posited that any act of prediction changes the brain state, making it impossible for the brain to predict its future state accurately. This implies that even if the brain operates mechanistically, it cannot predict its actions, supporting the concept of free will. Adolf Grünbaum (Grünbaum 1971) nuanced MacKay's claims, arguing that the inability of the brain to predict its actions does not necessarily impact the causal effects of decisions. Grünbaum introduced the concept of multiple realizability, suggesting that different physical states could result in the same decision, challenging the direct link between predictability and free will. The analysis in this chapter distinguishes between epistemic predictability (within a system) and ontological predictability (general predictability) especially regarding Laplace's demon and Popper's views on scientific determinism, emphasizing that ontological determinism is a broader and more general concept, not directly tied to empirical testability. In this chapter, Lloyd's Turing test for free is also analyzed, which proposes a test based on the Turing machine to argue for the existence of free will. He suggests that if a decision-making system can simulate its behavior, its decisions become unpredictable to itself, fostering a sense of free will. This argument rests on the system's ability to simulate its decision-making process and the intrinsic unpredictability of decisions. If Lloyd's arguments can be understood to pertain only to t-finite physical agents as presented in Chapter 5, then they can be validly used to explain why physical humans — and computers — can feel free.

Chapter 7 also uses the result in Chapter 5, that a computational prediction which is about the halting behavior of another computation within the same *t-finite* class is inherently unpredictable to explain the concept of qualia. According to Frank Jackson's knowledge argument (Jackson 2008), Mary, who lives in a black and white world, has all the physical knowledge about the world, yet she has new information when she sees a red apple. If we accept this, then physicalism — according to which the description of the world can be realized entirely with the help of physical theories — is false. However, even if Jackson's argument about the new information is true, we do not have to discard physicalism.

Suppose we interpret predictability in such a way that for an agent being unable to predict a specific fact of the world means that the information regarding that specific fact can't be known beforehand, but it can only be obtained either when the fact actually happens or at some later point of time. Then, Jackson's original argument can be understood as a problem regarding the predictive capacities of agents (Sóstai 2024). Based on the core argument of unpredictability, it can be shown even in the case of exclusively physical computer systems that these systems are not capable of predicting some of their own sensory information. First, a question related to sensation information is interpreted as a decision problem with the help of a formal model. After that, in the substantive part, it will be shown what kind of computational system structure and operating conditions are necessary for the creation of decision-related unpredictability. If the fulfillment of the operating conditions of such a system is required for the creation of qualia, then it can not be excluded that qualia can also be realized by a physical computer.

2. Key concepts

2.1. The Turing machine

An informal definition of a computation can be given as a process which follows a sequence of steps or rules ─ called an algorithm ─ to perform a task or solve a problem (Davis 1958, xvi). It involves systematically carrying out operations, which lead to a solution. An algorithm is a finite set of well defined instructions for performing a task which, when started with some initial information, will proceed through a series of steps or changes. These steps are designed to achieve a particular result and the algorithm will eventually come to a stop once this result has been reached. An algorithm is essentially a recipe for carrying out a specific process or solving a problem. The process can be done by hand, mechanically, or electronically by various systems, and it is fundamental to the workings of computers and many other types of devices.

A **Turing machine** *(TM)* is a conceptual model of computation invented by Alan Turing in 1936 (Turing, 1936) (Davis 1958, 5). It's a theoretical model which describes a device which manipulates symbols on a strip of tape according to a table of rules, by which it is able to compute any algorithm. Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm. A Turing machine is composed of a tape, a tape head, a state register, and a finite table of rules. A tape is divided into consecutive cells, each capable of holding a symbol. The tape is usually imagined as being infinitely long, allowing it to move left and right as far as needed for the computation. A tape head that can read and write symbols on the tape and move the tape left and right one cell at a time. A state register stores the state of the Turing machine. The number of different states is finite and each one is unique. A finite table of rules are the instructions that tell the machine what symbol to write and whether to move the tape left or right, or to halt, depending on the symbol it reads on the tape and the state it is currently in.

The machine manipulates symbols on a strip of tape according to a table of rules. To be more precise, it consists of a finite set of states $`q_1, q_2, q_3, \ldots`$, which are used to denote its internal configurations. It has a tape divided into squares, each of which may be printed with a symbol from a finite set $`S_0, S_1, S_2, \ldots`$. The machine performs operations according to a set of quadruples, which are expressions that specify its actions. These actions include printing a symbol, moving the tape left $(`L`)$ or right $(`R`)$, and transitioning to another state. Each **quadruple** (Davis 1958, 5) in the Turing machine is an instruction that, when the machine is in state $`q_i`$ and reads the symbol $`S_j`$ on the tape, causes it to perform a specified action and possibly transition to state $`q_k`$. A Turing machine is defined as a finite (nonempty) set of such quadruples, where no two quadruples have the same first two symbols, ensuring that each combination of state and tape symbol has at most one action associated with it.

An **input** (Davis 1958, 9) is the initial configuration of symbols on the Turing machine's tape before computation begins. It represents the data or problem that the machine is set up to process. For instance, if the Turing machine is designed to compute a function like addition, the input would be the numerical representations of the two numbers to be added, formatted according to the machine's symbolic system and placed on the tape for the machine to read. An **output** (Davis 1958, 9) is the resulting configuration of symbols on the Turing machine's tape after it has finished computing. This represents the solution to the problem or the result of processing the input data. Following the example with a Turing machine performing addition, after the addition the Turing machine would halt, and the sequence of symbols representing the sum of the input numbers would be the output, readable on the tape.

In a broader computational context an input is any data or instruction that is fed into a computational process or system. It can be in various forms, such as keyboard strokes, mouse clicks, sensor readings, or data read from files or over a network. An output is any data or information that is produced by a computational process or system. It can be presented in numerous ways, such as visual display on a screen, printed material, sound, signals to control machines, or data written to a file or sent over a network.

The Turing machine operates by beginning in a specified initial state with an initial tape configuration and proceeds step by step, following the rules defined by its quadruples, until it reaches a configuration for which no rules apply, at which point it halts.

The transition function (Sipser 2006, 167) of a TM is the total set of instructions or quadruples that dictates the machine's next action based on its current state and the symbol it reads from the tape. For a given state and symbol, the transition function specifies the symbol to write to the tape, the direction to move the tape head, and the subsequent state to transition into. The transition function can otherwise be called as the program of the Turing machine.

An instantaneous description (Davis 1958, 6) of a TM captures a snapshot of the Turing machine's computation at a particular moment. It includes the current state of the machine (denoted by q_i), the current content of the tape (including the symbol currently under the head), and the position of the head. Therefore, an instantaneous description gives us all the necessary information to know the current configuration of the Turing machine and to determine its next move according to its transition function. It's an abstraction of the machine's 'moment in time', crucial for understanding its operation and predicting its future behavior. So it is basically a description of where the machine is at 'right now'.

As an example of an instantaneous description α_0, let's consider a simple Turing machine working with the following quadruple (q_0, A, B, q_1). This quadruple tells the Turing machine that if it is in state q_0 and reads the symbol $'A'$ on the tape, it should print $'B'$ in place of $'A'$ and move into state q_1. Suppose the tape initially looks like this, with the head reading the first $'A'$:

$..., X, A, Y, Z, ...$
$..., \ , q_0, \ , \ , ...$

The instantaneous description α_0 at this point would be $..., X, \underline{A}^{q0}, Y, Z, ...$, where $\underline{A}^{q0}$ indicates the machine is in state q_0 and the head is reading $'A'$. After reading and processing the quadruple following quadruple (q_0, A, B, q_1) and assuming the machine moves right (the direction is not specified in this quadruple, so let's assume a right move for this example), the tape and machine would look like the following:

$..., X, B, Y, Z, ...$
$..., \ , \ , q_1, \ , ...$

The instantaneous description α_1 at this point would be $..., X, B, \underline{Y}^{q1}, Z, ...$, which means the machine has moved into state q_1 and the head has moved to read $'Y'$.

Suppose we have T as a Turing machine and we have α as an instantaneous description of T, where q_i is the internal configuration that occurs in α and where S_j is the symbol immediately to the right of

q_i. Then, **the internal configuration** of a Turing machine (Davis 1958, 6) refers to its current state. It is symbolized by q_i and is part of the instantaneous description of the machine. It represents the specific condition or control state of the TM at a particular moment during its computation. **A terminal or halting instantaneous description** (Davis 1958, 7) is a condition where no further moves can be made according to the machine's transition function, which is indicated as α with respect to TM_T. It's called 'terminal' with respect to T if there are no rules in T that define what the machine should do next. In simpler terms, the machine has reached a condition where it cannot continue processing based on its instructions; it has halted.

A computation of a Turing machine (Davis 1958, 7) is a sequence of steps (instantaneous descriptions), starting with the initial configuration and proceeding through successive states as directed by the transition function. Each step α_i leads to the next step α_{i+1}, for $1 \leq i < p$, where α_i is the initial configuration. The computation is considered complete when a terminal instantaneous description α_p is reached. The term 'Resultant of α_1 with respect to T' (denoted as $\text{Res}_T(\alpha_1)$) refers to the output or final state of the Turing machine after the computation ends. More simply, a computation is a series of instantaneous descriptions where from one such description we obtain the next one with the help of the transition function.

So, in summary, the internal configuration is the current state of the TM, a terminal instantaneous description signifies the end of the computation process where the TM can no longer proceed, and computation is the process itself, described as the sequence of conditions the TM transitions through from the beginning to the terminal or halting condition.

2.2. Turing machines performing numerical computations

A Turing machine can perform numerical computations (Davis 1958, 9) by using a symbolic representation for numbers on its tape in the following way. A specific symbol S_1 is chosen to represent the number 1. A positive integer n is then represented on the tape as a sequence of n occurrences of the symbol S_1. This is written as S_1^n, which stands for the expression $S_1S_1...S_1$ (n times). For the number 0, a null expression is used, which can be represented as S_1^0 and implies that no symbols are written on the tape for this number.

Therefore, to perform numerical computations, a Turing machine will read and manipulate sequences of these symbols according to its transition function. For example, to add two numbers, the machine would have sequences representing each number on the tape and would be programmed to combine these sequences, resulting in a sequence that represents their sum. The numeral *'n'* is associated with the tape expression $11...1$ with $n+1$ occurrences of the symbol for 1, signifying the unary representation of numbers in the Turing machine model. Using $n+1$ ones ensures that there is a clear, non-ambiguous representation for each integer, which includes zero. Zero is represented as a single 1, distinguishing it from an empty sequence, which might be used to represent the absence of a number or an uninitialized portion of the tape. This way, every non-negative integer has a unique, non-empty representation.

2.3. Turing machines computing functions

We've established that Turing machines can perform numerical computations by representing numbers symbolically on the tape and manipulating these representations according to the rules of the transition function. Computing a function f is an extension of this capability.

Let's say we have a Turing machine designed to compute a function f, such as addition. To compute $f(x, y) = x + y$, we would represent the numbers x and y symbolically on the tape (for example 111 for the number 2 if using unary representation). The Turing machine would then follow its transition function to manipulate these representations and produce the sum $x + y$ on the tape.

So, the ability of a Turing machine to compute a function is based on its ability to perform numerical computations. The function itself is an abstraction of numerical operations that the machine can realize through a sequence of basic steps defined by its transition function. Whether it's a partial or total function, the machine's computational process involves reading, writing, and changing states as it processes the input symbols (which represent numerical values) on the tape.

A function $f(x_1, ..., x_n)$ is partially computable if there exists a Turing machine T that, given an input consisting of n numerical values represented on its tape, processes these values according to the rules defined by its transition function and produces an output on the tape that corresponds to the value of the function f for those inputs. The output of the Turing machine T when given the input $(x_1, ..., x_n)$ is written as $\Psi_T^{(n)}(x_1, ..., x_n)$, which signifies the result of the computation by machine T for the function f with n arguments.

A function is partially computable when the Turing machine may not halt for some inputs, meaning it does not produce an output for some inputs. If the function f is defined for all possible inputs and there is a Turing machine T that halts with the correct output for every possible input, then f is not just partially computable, but computable.

2.4. Effectively calculable functions

An effectively calculable function is one that a Turing machine can compute in a finite number of steps. If a function $f(m)$ is computable by a Turing machine T, we can start with an initial number m_0 and create an initial instantaneous description α_1. The Turing machine then successively generates instantaneous descriptions $\alpha_1 \rightarrow \alpha_2 \rightarrow ... \rightarrow \alpha_p$, where α_p is terminal, so it represents a halting state of the machine. Since $f(m)$ is computable, α_p must be reached after a finite number of steps, and $f(m_0)$ is defined to be the number represented by the terminal description α_p.

A computable function must be effectively calculable by definition (Davis 1958, 11) because the Turing machine must reach the final output with a terminal or halting state in a finite number of steps. This means that there is a guaranteed end to the process, and the result can be known after a deterministic and bounded amount of computation. It's effectively calculable because the process by which the computation is carried out is systematic, predictable, and can be completed without the need for infinite resources or time. In short, the calculation of the function is effective because the computation halts with a correct output.

2.5. Relative computations, subroutines

Relative computation (Davis 1958, 20-22) allows a Turing machine to use another entity — which could be another Turing machine — as a subroutine. This is conceptualized through the ability of the Turing machine to make external queries to a set A, which can be thought of as representing the answers provided by another process or Turing machine. In the traditional Turing machine model, the machine operates in a closed system with a finite set of states and transitions, and all possible actions it can take are determined by its internal configuration and the symbols it reads on its tape. It cannot go beyond this to access external information or computations. However, we can extend this model to include relative computability by allowing the Turing machine to reference an external source of information. This source could very well be the output of another Turing machine.

An A-computation, relative computation or a computation of a subroutine of a Turing machine T involves a sequence of steps (instantaneous descriptions) where the Turing machine may interact with an external source (represented by T) through queries related to the set A. These queries are effectively asking whether certain elements are in the set A, which is external to the Turing machine itself. The computation sequence involves a series of transitions from one instantaneous description to the next, where each transition is potentially influenced by the external information from T concerning membership in A. This interaction is denoted by $\alpha_1 \rightarrow \alpha_1\ (T)$ for transitions that do not depend on membership in A and $\alpha_1 \ \ {}^{A}\!\rightarrow \alpha_1\ (T)$ for transitions that do depend on whether a certain condition related to A is true or not.

The presence of these A-dependent transitions is what makes this computation relative. The Turing machine's behavior at certain steps depends on external information rather than being determined solely by a predefined set of rules or states. The computation is said to be complete when it reaches a final instantaneous description α_p beyond which no further transitions are defined or required. This α_p is known as the A-resultant of the initial state α_1 with respect to the external source T. This represents the outcome of the Turing machine's computation after possibly many interactions with the external source.

2.6. Turing machines computing solutions to decision problems

A (decision) problem UP can be defined as a class of questions Q (Crossley 1990, 37) related to a specific domain D. This domain D represents the set of all possible instances or inputs for which the problem UP seeks to determine a particular property or answer a specific question. Domain D is the set of all items or instances that are relevant to the problem. For example, if UP is *'determining primality'*, then D would be the set of all natural numbers. The class of questions Q is the set of questions that can be asked about elements of D for which the problem UP seeks answers. In the primality example, each question q_i in Q would take the form *'Is the number m a prime number?'* Each question q_i in Q seeks to establish whether a certain property holds for an instance in D. The property corresponds to the solution of the problem UP for that instance.

A decision problem UP is characterized by the ability to frame every instance d_i in the domain D as a question q_i in Q. The problem UP is effectively a request for a method or procedure that can consistently and correctly determine the answers to all questions in Q. When the problem UP is associated with a decision problem, it is concerned with whether elements in D have a certain binary property, such as *'prime vs. non-prime.'* In this case, a problem can be solved if there is a procedure (a Turing machine) that, for every question q_i derived from D produces a correct yes/no answer (or equivalently, $1/0$ in the case of a characteristic function). Therefore, a problem in the context of computability theory is the task of finding a computational method to determine a specific property for all possible instances within a certain domain, which can be expressed as a class of yes/no questions.

When we focus on decision problems, then a problem as class of questions Q is said to be decidable if and only if there exists a total Turing machine T such that for every question q_i in Q, the following two conditions are met:

1. T is a total Turing machine, which means that for every input q_i, T halts after a finite number of steps.

2. T, when given any question q_i from the class Q, will produce an output that corresponds to a correct answer:

- The machine T halts with output 1 if the answer to q_i is $'YES'$.
- The machine T halts with output 0 if the answer to q_i is $'NO'$.

The class Q is effectively decidable by T because there exists an algorithmic method, implemented by T, that can resolve every instance q_i within the class in a finite amount of time. The key point here is that the Turing machine T operates uniformly across the entire class Q, demonstrating the class's decidability by providing a consistent decision-making process for every possible question q_i it contains.

2.7. Example

Using the example *'is m a prime number?'*, we can explain the formal definition of decidability in terms of a total Turing machine. Let Q be the class of questions of the form *'Is m a prime number?'*, where each question Q_i in Q corresponds to a specific natural number m. According to the formal definition of decidability, we have to show that there exists a total Turing machine T which can decide whether any given number m is a prime. A total computable Turing machine T is constructed such that it always halts after a finite number of steps for any given natural number m. This is possible because the primality test for any specific number is a procedure that can be completed in finite time; there are known algorithms for this. T can be defined as follows:

If m is a prime number, then T processes this input and eventually halts with the output 1, indicating a $'YES'$ answer.

If m is not a prime number, then T also halts after a finite number of steps but with the output 0, indicating a $'NO'$ answer.

In the context of the class of questions Q, the Turing machine T functions as an algorithm that uniformly applies a primality test to any given m. Since T halts with the correct answer for every possible m (whether m is a prime number or not), we can say that the class Q is decidable. This is a demonstration that the property of being a prime number is one that can be algorithmically determined for all natural numbers, therefore it is a decidable class of questions.

2.8. Relations between sets, function, programs and decision problems

Computability theory investigates the capabilities and limitations of computers in solving problems. (Davis 1958, 66-78) It categorizes problems based on their inherent difficulty and the computational resources required to solve them. We can give a short definition of all inter–related concepts in order to show their relationships.

Turing machines or Programs

A program P is total (computable) if for every input x, there exists a Turing machine T such that $T(x)$ halts and produces an output y in a finite number of steps. A total program is a type of computable program that terminates and produces an output for every possible input. In other words, it is a program for which a corresponding Turing machine halts on all inputs, making it synonymous with *'computable'* in the context of functions or programs. A program P is partially computable if for some inputs x, there exists a Turing machine T such that $T(x)$ halts and produces an output y in a finite number of steps, and for other inputs, $T(x)$ may run indefinitely without halting. A partial program can produce a correct output for some inputs within finite time but may not halt for others. A program P is uncomputable if there is no Turing machine T such that for every input x, $T(x)$ halts and produces an output in a finite number of steps.

Computable programs are a subset of partially computable programs. Every computable program is partially computable because it halts and produces output for every input, fulfilling the criteria for partial computability. Uncomputable programs do not overlap with computable or partially computable programs, representing problems or functions no Turing machine can solve in finite time for all inputs.

Decision Problems

A decision problem UP is semidecidable if there exists a Turing machine T such that for any input x where x is in the domain D, $T(x)$ halts and outputs *'yes.'* For any input x not in D, $T(x)$ may not halt at all. The key characteristic is that it must halt and output *'yes'* for all x in D. A decision problem UP is decidable if there exists a Turing machine T such that for any input x, $T(x)$ halts and correctly outputs *'yes'* if x is in D and *'no'* if x is not in D, in a finite number of steps. A decision problem UP is undecidable if there is no Turing machine T that, for every input x, halts and correctly decides whether x is in D in a finite number of steps.

Decidable problems are a subset of semidecidable problems. If a problem is decidable, then it is also semidecidable because a Turing machine that halts with answers for both *'yes'* and *'no" instances implicitly halts for* 'yes' instances. Undecidable problems are those that cannot be resolved into either decidable or semidecidable problems due to the absence of a Turing machine that can universally solve them.

Sets

A set S is recursively enumerable if there exists a Turing machine M that enumerates the members of S. Formally, S is recursively enumerable if there exists a partial computable function f such that $S = dom(f)$, the domain of f. A set S is recursive if there exists a Turing machine M that decides membership in S. Formally, S is recursive if there exists a total computable function f such that for every element x, $f(x) = 1$ if $\boldsymbol{x} \in \boldsymbol{S}$ and $f(x) = 0$ if $\boldsymbol{x} \notin \boldsymbol{S}$. A set S is co-REcursively enumerable if its complement $\check{S}$ (the set of all elements not in S) is recursively enumerable. This means there exists a Turing machine M' that halts and accepts x for every $\boldsymbol{x} \notin \boldsymbol{S}$. A set S is non-recursively enumerable if there is no Turing machine M that can enumerate all and only the members of S. Formally, S is non-recursively enumerable if there is no partial computable function f with S as its domain.

Recursive sets are both RE and co-RE. They represent the intersection of RE and co-RE sets. This is because a recursive set has a Turing machine that decides membership for every element, implying both the set and its complement can be enumerated by halting Turing machines.These are the most computationally tractable sets, having clear criteria for membership and non-membership, decidable by a Turing machine. A set and its complement are RE and co-RE, respectively. If both a set S and its complement $\check{S}$ are RE, then S is actually recursive. This symmetry is crucial for understanding the boundary between decidable and undecidable problems. RE and co-RE are less tractable. For RE sets, membership can be proven by a Turing machine that halts, but non-membership may not be provable. For co-RE sets, non-membership can be proven by halting, but membership may not be provable. Non-RE sets fall outside the realm of RE and co-RE. They represent the most challenging category, where algorithmic methods cannot enumerate the sets or their complements effectively.

Functions

A function f is computable or recursive if there exists a Turing machine M such that for every input x, $M(x)$ halts and produces $f(x)$ in a finite number of steps. A function f is partially computable or partial recursive if it is defined (halts and produces an output) for some inputs but not for others. This means there exists a Turing machine M that computes $f(x)$ for inputs where f is defined. A function f is uncomputable if there is no Turing machine M that, for every input x, halts and produces $f(x)$ in a finite number of steps.

Computable functions or total functions are partially computable functions because they define an output for every input. The domain of computable functions is included within partially computable functions. Uncomputable functions are those functions for which no Turing machine can produce outputs for all inputs in finite time. These are distinct from both computable and partially computable functions.

The terms *'effectively calculable function'* and *'computable function'* can be used interchangeably in the context of computer science and mathematics. The concept of a function being *'effectively calculable'* means that there exists a systematic procedure for computing the function's value for any valid input in a finite amount of time.

Equivalence of Partial Recursive Functions and Turing-computable Functions

The equivalence of partial recursive functions and the functions computable by Turing machines is foundational to computability theory and is often treated as an axiom or a definition rather than something that is proven. The justification for this equivalence comes from the Church–Turing thesis and can be understood through the construction of Turing machines that compute these functions and vice versa. If it is accepted that all computations can be done by Turing machines then the computable partial functions are exactly the partial recursive functions.

A partial recursive function relies on basic operations such as substitution and primitive recursion. Each of these operations can be precisely executed by a Turing machine designed for that purpose. The construction of such machines demonstrates how they can perform any operation required by partial recursive functions. Moreover, any function computable by a Turing machine can be described using the operations that define partial recursive functions. This is because the finite set of instructions in a Turing machine's program, which computes the function and determines its halt or indefinite run, can be mirrored by recursive function operations. The equivalence is therefore justified through construction: showing that for every operation that defines partial recursive functions, there's a corresponding Turing machine, and for every computation a Turing machine can perform, there's a corresponding description using the operations that define partial recursive functions.

2.9. Examples

We can use the example of prime numbers to illustrate each concept across Turing machines or programs, decision problems, sets, and functions.

Decision Problem $UP1$: *'is x a prime number?'*

Decidable problem: $UP1$ is decidable because there exists a Turing machine $T1$ which, for any input x, determines whether x is prime and produces an output y (*'yes'* for prime, *'no'* for not prime) in finite steps. This illustrates the property of being decidable; there's an algorithm (primality test) that solves it for any given x efficiently.

Corresponding Turing machine or program: The program that implements $T1$ for checking primality is a total (computable) program. Given any integer x, it always halts with a definitive answer in a finite number of steps, embodying the essence of computability.

Corresponding set: The set $S1$ *'The set of all prime numbers'* is recursive because there exists a Turing machine (which can be the same as $T1$) that decides membership: for any given x, it determines in finite time whether x belongs to $S1$ (is prime) or not, aligning with the concept of a decidable problem.

Corresponding function: The function $F1$ *'The characteristic function for prime numbers'* — $F1(x) = 1$ if x is prime, and $F1(x) = 0$ otherwise. This function is computable because there exists a Turing machine (implementing $T1$) that, for any input x, calculates $F1(x)$ in finite time, thus demonstrating the property of being a total function.

In order to show the concepts of partially computable programs, semidecidable problems, recursively enumerable sets, and partially computable functions with an example, we need a problem that has an algorithm that halts for *'yes'* instances but may not halt for *'no'* instances, which is not directly applicable using the prime number example as it's fully decidable. These concepts are aligned with

problems like the halting problem, where the Turing machine halts for programs that do halt (confirming their halting) but may run indefinitely for programs that don't halt, therefore it does not provide a 'no' answer effectively.

Decision Problem $UP2$: *'Does Turing machine P halt on input x?'*

The halting problem (Crossley 1990, 38–40)(Davis 1958, 70–71)(Turing 1936) is semidecidable because there exists a Turing machine H that, for any input pair (P, x) where Turing-machine P halts on input x, H halts and outputs *'yes.'* However, if P does not halt on x, H may run indefinitely, making it impossible to always get a definitive *'no'* answer. This illustrates the property of being semidecidable: there's an algorithm that resolves 'yes' instances but may not halt for 'no' instances.

Corresponding Turing machine or program: A program implementing H for the halting problem is partially computable. It can determine and halt with a *'yes'* answer for pairs (P, x) where P halts on x, but it does not halt for pairs where P does not halt on x, showcasing partial computability.

Corresponding set: *'The set S2 of all pairs (P, x) such that Turing machine P halts on input x'* is recursively enumerable because there exists a Turing machine (the same H) that can enumerate all pairs (P, x) for which P halts on x. However, there's no mechanism to enumerate pairs where P does not halt on x, reflecting the essence of semidecidability and recursively enumerable sets.

Corresponding function: *'A function that outputs 1 if Turing machine P halts on input x, and is undefined otherwise.'* – $F2(P, x) = 1$ if P halts on x, and $F2(P, x)$ is undefined if P does not halt on x. This function is partially computable because there exists a Turing machine (implementing H) that computes $F2(P, x)$ for halting instances but may run indefinitely for non-halting instances, underscoring the property of partial computability.

By considering the complementary aspect of the halting problem – focusing on non-halting programs – it's possible to illustrate uncomputable programs, undecidable problems, non-recursively enumerable sets, and uncomputable functions.

Decision Problem $UP3$: *'Does Turing machine P never halt on input x?'*

Undecidable Problem: This problem is undecidable (Sipser 2006, 207) because there is no Turing machine N that can, for every input pair (P, x), correctly decide if Turing-machine P does not halt on input x (outputs *'yes'* for never halting, *'no'* for halting) in finite time. The challenge arises because proving non-halting requires exhaustive exploration of all possible computational paths, a task that is not feasible within finite time for all possible P and x.

Corresponding Turing machine or program: A program intended to solve $UP3$ for any arbitrary Turing machine P and input x would be uncomputable. No such program can exist that always halts with a correct answer, because it would need to solve the undecidable problem of determining non-halting, which is inherently uncomputable.

Corresponding set: The set $S3$ *'The set of all pairs (P, x) such that Turing machine P does not halt on input x'* is non-recursively enumerable (Sipser 2006, 207–208) because there is no Turing machine that can enumerate all and only the pairs where Turing-machine P does not halt on input x. The

inability to algorithmically confirm non-halting instances for all P and x in finite time means no exhaustive listing mechanism exists for this set, distinguishing it as non-recursively enumerable.

Corresponding Function: The function $F3$ *'A function that outputs* 1 *if Turing machine* P *never halts on input* x*, and is undefined or outputs* 0 *otherwise.'* is uncomputable because determining its value requires solving $D3$, which is undecidable. There's no Turing machine that can compute $F3(P, x)$ for every possible P and x, specifically due to the impossibility of algorithmically confirming non-halting across all computational paths in finite time.

2.10. Encoding turing machines

The process of encoding (Davis 1958, 56-62) a Turing machine is called Gödel-numbering. This arithmetization means a transformation of abstract computational descriptions into concrete numerical representations. Here, we focus on a simplified TM model, where the tape symbols are limited to $'1'$ and blanks, and the machine's instructions are composed of quintuples instead of quadruples for simplicity and to avoid ambiguity, but could also be done for quadruples (Davis 1958, 56). The encoding process unfolds in three key stages, ultimately yielding a unique number through the application of the Fundamental Theorem of Arithmetic, which asserts that every positive integer greater than 1 is uniquely decomposable into prime factors.

Stage 1: Encoding a Turing machine to a Sequence of Symbols

A one-symbol Turing machine operates with a simplified set of actions due to its restricted alphabet. Each instruction or quintuple in the machine's set can be represented as $(q_i, 1, B, d_j, q_k)$ or $(q_i, B, 1, d_j, q_k)$, where:

– q_i and q_k are the current and next states, respectively,
– $'1'$ represents the symbol $'1'$ on the tape, and $'B'$ represents a blank,
– d_j indicates the direction in which the tape head moves $(Left, Right\ or\ Stays)$.

We consider Turing machines that share the same program to be identical. Therefore, we can describe a Turing machine program as a sequence of command lines, with each command line itself being a sequence of symbols. To encode a Turing machine as a sequence of symbols, we can simply concatenate its command lines into a single continuous string. This way it's possible to encode the machine's set of quadruples into a sequence of symbols that uniquely represents an instruction. Suppose we have the following Turing machine program using two set of quintuples:

$0, B, 1, R, 1$
$1, 1, B, L, 2$

To encode this as a sequence of symbols, we simply have $0, B, 1, R, 1, 1, 1, B, L, 2$.

Stage 2: Encoding a Sequence of Symbols to Sequence of Numbers

Each symbol from stage 1 is mapped to a unique natural number. This creates a sequence of numbers that reflects the original sequence of symbols. Each quintuple in a Turing machine program results from filling out the following parameters: (current state); (current symbol); (new symbol) or

(direction); (new state). The parameters (current state) and (new state) are filled in with numerals. We can code each numeral using the number it represents:
“0” → 0
“1” → 1
“2” → 2
…

Since we’re working with one-symbol Turing machines, the parameters (current symbol) and (new symbol) are filled in with “1” or “B”. The encoding is done the following way:

“B” → 0
“1” → 1

The parameter (direction) is filled with “R”, “L”, or “ * ”, so the encoding is done the following way:

“R” → 0
“L” → 1
“*” → 2

So the former sequence of symbols *0, B, 1, R, 1, 1, 1, B, L, 2* is encoded to a sequence of numbers: *0, 0, 1, 0, 1, 1, 1, 0, 1, 2*.

Stage 3: Sequence of Numbers to Unique Number

The final stage involves transforming the sequence of numbers obtained from stage 2 into a single, unique number that encodes the entire Turing machine. Given that each instruction and element within an instruction is represented by a unique prime raised to a unique power (reflecting its position and symbol), the resulting product is a number that uniquely encodes the entire TM.

We can code the sequence of numerals $(n_1, n_2, \ldots, n_k)$ as the number $p_1^{n1+1} \times p_2^{n2+1} \times \ldots \times p_k^{nk+1}$, where p_i represents the i-th prime number. By adding one to exponents, we can ensure that the coding scheme is unambiguous. Otherwise we would use 1 as a code for each of the following sequences: *(0), (0, 0), (0, 0, 0),* etc. As an example of this encoding, the sequence of numbers *(0,0,1,2,2)* would be encoded the following way: $2^{0+1} \times 3^{0+1} \times 5^{1+1} \times 7^{2+1} \times 11^{2+1} = 2 \times 3 \times 25 \times 343 \times 1331 = 68\,479\,950$

A fundamental theorem of arithmetic ensures that this encoding is reversible. From the unique number we can uniquely factorize it back into the prime factors and their exponents. This way we can reconstruct the original sequence of symbols and, consequently, the set of quintuples that defined the Turing machine.

Example

Given our encoding scheme, we can calculate the unique Gödel-number of the former sequence of symbols *0, 0, 1, 0, 1, 1, 1, 0, 1, 2* :

$2^{0+1} \times 3^{0+1} \times 5^{1+1} \times 7^{0+1} \times 11^{1+1} \times 13^{1+1} \times 17^{1+1} \times 19^{0+1} \times 23^{1+1} \times 29^{2+1} = 2 \times 3 \times 25 \times 7 \times 121 \times 169 \times 289 \times 19 \times 529 \times 24389 = 1\,521\,116\,521\,577\,602\,950$

This number, based on the Fundamental Theorem of Arithmetic, has a unique prime factorization, which ensures that our encoding is unambiguous and can be decoded back to the original Turing machine operation if needed.

2.11. The universal Turing machine

A Universal Turing machine (UTM) (Crossley 1990, 39-41)(Davis 1958, 64-65)(Turing 1936) is a Turing machine that can simulate any other Turing machine. It's capable of performing any computation that any other Turing machine can, given the correct input and sufficient time. The Universal Turing machine operates by reading the description and state of another machine, along with the input for that machine, and then executing the steps that the other machine would take. This is accomplished by representing the other Turing machine's transition function and initial tape configuration within the input of the universal Turing machine. When the universal Turing machine processes this input, it basically *'runs'* the other machine. This ability to simulate any Turing machine makes the Universal Turing machine a powerful model for computation, as it can perform any computation that can be algorithmically defined.

Consider a Turing machine U that takes as its input a pair (z, x), where z is the Gödel number and x is the input on which the encoded Turing machine is supposed to run. The universal Turing machine U is designed such that it computes a function $\Psi_U(z, x)$, which is equal to the function $\Psi_Z(x)$ computed by the Turing machine encoded by z on the input x. The universality comes from the machine's ability to compute any partially computable singular function, meaning it can handle any function that a Turing machine can compute, given that it terminates or halts on that input. To use U to compute an n–ary function, we can encode the n inputs into a single number using a predefined scheme, and then run U on this encoded number. The UTM reflects the principle that a single system can model the computation of any algorithm, as long as that algorithm can be encoded as a sequence of symbols (the Gödel number) and is computable (meaning the process eventually halts and provides an answer).

2.12. The halting problem

Now suppose the following case (Rayo 2019, 238-239), where $UD2$ was the problem or the class of questions: *'Does the Turing machine P halt on some input i?'*

Definition of Program H

There exists a set of input pairs $(\text{'}P\text{'}, i)$ that H can process, each input is a variable representing a specific program P and its specific input i. Input $(\text{'}P\text{'}, i)$ is a pair consisting of a description or Gödel-number based encoding of a specific instance of program P and an input i. Each P represents a unique program or computational process in a standard enumeration of programs (Davis 1958, 56-62), and each i represents a specific input in a standard enumeration of inputs to that program P. To each input of H, there corresponds a set of outputs o that H produces, indicating whether each program P halts on its specific input i. Output o is a binary value, 1 or 0, indicating whether the program P halts on input i (1 if $P(i)$ halts, 0 if $P(i)$ does not halt). For each input pair $(\text{'}P\text{'}, i)$, the output o systematically indicates the halting behavior of P on i, accurately reflecting whether or not P terminates. The following proof, whose first version was given by Turing in 1936 (Turing 1936) proves that the halting problem is uncomputable, because there exists at least one program whose halting status can't be computed by program H.

H represents the problem $UD2$ if for all instances P and i:

If $P(i)$ halts, then $H(\text{‘}P\text{’}, i) = 1$.
If $P(i)$ does not halt, then $H(\text{‘}P\text{’}, i) = 0$.

Program H execution:

For any given input pair $(\text{‘}P\text{’}, i)$:

H evaluates whether P halts on i.
H Outputs 1 if P halts, accurately reflecting a termination state.
H Outputs 0 if P does not halt, accurately reflecting an ongoing computation without termination.

We can consider the following example of a situation by considering it a variant of the undecidability of the halting problem. Here is a variant of how the paradox can be instantiated.

We can define the following anti-diagonal program D. D's program is special, because what it does is the following sequence of actions in sequential order. First, D takes one input i, then computes $H(\text{‘}I\text{’}, i)$ as a subroutine, where $\text{‘}I\text{’}$ stands for the i-th specific program in a Gödel-number based enumeration of Turing-machine. Such a computation and using H as a subroutine is possible, it is detailed in the former definition of relative computation. Second, $H(\text{‘}I\text{’}, i)$ returns an output value (0 or 1). Then D uses that output value in the following way:

If $H(\text{‘}I\text{’}, i) = 1$, then $D(i)$ doesn't halt.
If $H(\text{‘}I\text{’}, i) = 0$, then $D(i)$ halts.

Program D on input $\text{‘}D\text{’}$ – contradicts the output of $H(\text{‘}I\text{’}, i)$:

If $H(\text{‘}D\text{’}, \text{‘}D\text{’}) = 1$, then $D(\text{‘}D\text{’})$ doesn't halt.
If $H(\text{‘}D\text{’}, \text{‘}D\text{’}) = 0$, then $D(\text{‘}D\text{’})$ halts.

This results in the following contradiction(s):

If $D(\text{‘}D\text{’})$ halts, then $H(\text{‘}D\text{’}, \text{‘}D\text{’}) = 1$.
If $H(\text{‘}D\text{’}, \text{‘}D\text{’}) = 1$, then $D(\text{‘}D\text{’})$ doesn't halt.

If $D(\text{‘}D\text{’})$ doesn't halt, then $H(\text{‘}D\text{’}, \text{‘}D\text{’}) = 0$.
If $H(\text{‘}D\text{’}, \text{‘}D\text{’}) = 0$, then $D(\text{‘}D\text{’})$ halts.

A variant $UD4$ of the halting problem and the corresponding diagonal argument (Crossley 1990, 38-40) can also be stated as follows: *'Does the Turing machine M on input $\text{‘}M\text{’}$ halt?'*

Definition of Program S

There exists a set of input pairs $('M', 'M')$ that S can process. Each input is a variable representing a specific program M and its specific input 'M', in which case 'M' is the Gödel-number of the Turing machine M. Each M represents a unique program or computational process in a standard enumeration of programs. To each input of S, there corresponds a set of outputs o that S produces, indicating whether each program M halts on its specific input $'M'$. Output o is a binary value, 1 or 0, indicating whether the program M halts on input $'M'$ (1 if $M('M')$ halts, 0 if $M('M')$ does not halt). The following proof proves that the variant of the halting problem is uncomputable, because it leads to a contradiction.

S represents the problem $UD4$ if for all instances M and $'M'$:

If $M('M')$ halts, then $S('M', 'M') = 1$.
If $M('M')$ does not halt, then $S('M', 'M') = 0$.

Program S execution:

For any given input pair $M('M')$:

S evaluates whether M halts on $'M'$.
S Outputs 1 if M halts, accurately reflecting a termination state.
S Outputs 0 if M does not halt, accurately reflecting an ongoing computation without termination.

We can consider the following example of a situation by considering it a variant of the undecidability of the halting problem, when S tries to determine if $S('S')$ halts. Here is a variant of how the paradox can be instantiated:

Program S on input 'S' ─ contradicts the output of $S('S', 'S')$:

If $S('S')$ halts, then $S('S', 'S') = 1$.
If $S('S')$ does not halt, then $S('S', 'S') = 0$.

One of the immediate implications of the halting problem is that there is no general computable method to determine whether an arbitrary program (or Turing machine) will halt for every possible input. This directly implies that there is no computable method to filter out partial functions (functions that are not defined for every possible input in their domain) from total functions (functions that are defined for every input in their domain) when these functions are represented as programs or Turing machines. Why this poses a more deeper problem can be understood once we can establish its consequences by connecting it to the Church─Turing thesis first.

2.13. The Church─Turing thesis

The Church─Turing (CTT) thesis is a fundamental concept in computer science that relates to the nature of computation and what can be computed by machines (Copeland 2020). It asserts that any function that can be effectively computed can be computed by a Turing machine. More precisely it can

be stated as: *'A function f_e is effectively computable if for every input i, a Turing machine program P on input i correctly halts within finite steps'.* The thesis is named after Alonzo Church and Alan Turing. Church developed the concept of lambda calculus, while Turing introduced the Turing machine, both in the 1930s. Despite the differences in their approaches, both models were proven to be equivalent in terms of computational power. The lambda calculus ─ developed by Alonzo Church ─ is a formal system for expressing computation based on function abstraction and application.

The Church─Turing thesis asserts that if a function is computable in any model that satisfies certain basic computational principles, then it is computable by a Turing machine, and vice versa. This leads to the concept of *'Turing completeness'*, which means a system that can simulate a Turing machine is capable of performing any computation that a Turing machine can, if it is given sufficient time and resources.

There are many aspects of the Church─Turing thesis, which can be the source of investigation. The CTT is not a formal mathematical statement but rather a hypothesis about the nature of computation. It asserts that any function that can be effectively computed can be computed by a Turing machine or an equivalent model of computation. The thesis is not entirely empirical, as it is based on the intuitive notion of what constitutes an effective procedure, which is then formalized through the concept of Turing machines. The essence of the CTT is its attempt to formally articulate what constitutes an *'effective procedure'* for computation.

2.14. Effective procedures

In the disciplines of logic, mathematics, and computer science, the term *'effective'* ─ along with its synonyms *'systematic'* and *'mechanical'* ─ carries a specialized meaning that diverges from their everyday usage (Copeland 2020)(Piccinini 2015, 5). These terms are employed to describe methods or procedures characterized by a set of distinct properties that distinguish them as effective. An effective method must be done through a finite series of precise instructions. Each of these instructions must be expressible by a finite number of symbols, ensuring that the method can be communicated and understood. When executed correctly, an effective method is certain to yield the desired outcome within a finite number of steps. This predictability and reliability are crucial for a method to be considered effective. The procedure should be executable by a human without the aid of any machinery, aside from basic tools such as paper and pencil. This criterion emphasizes the method's accessibility and the universality of its applicability, regardless of technological assistance. Executing an effective method requires no special insight, intuition, or ingenuity from the individual performing it. The method relies solely on the mechanical application of its instructions, making it accessible to anyone who follows the prescribed steps.

An example of such an effective method is the truth-table test for determining the tautologous nature of propositions in propositional calculus. This method embodies the characteristics of effectiveness, as it is systematically set out in a finite number of exact instructions that, when followed meticulously, assure the achievement of the desired result. In principle, any human can carry out this test by sheer rote, provided they have enough time, persistence, paper, and pencils. Although, in practical terms, this method becomes infeasible for formulas containing a significant number of propositional variables due to the exponential growth in complexity.

From the perspective of the halting problem, a computation to be considered effective, it must satisfy two fundamental criteria:

It must halt: This means that the computation finishes in a finite amount of time for any given input. A computation that never halts for some inputs is not considered effective because it does not produce an outcome that can be used.

It must produce a correct result: The computation must correctly solve the problem it was designed to address, according to the specifications or the intended outcome. This involves producing accurate, valid results that meet the problem*'s requirements or the computation'*s goals.

The requirement for halting ensures that a computation reaches a conclusion, making its outcome accessible for evaluation or use. The correctness requirement ensures that this outcome is not just any result, but the right one, fulfilling the purpose for which the computation was undertaken. There is also a deeper realization, which is important when we're connecting the Church–Turing thesis to the halting problem. And that is that **for a computation that does not halt, it's not possible to determine its correctness** – if by correctness we mean that it solves or decides the problem it was designed for. In order to understand this, we should also connect the notion of effectiveness to the decidability of a problem. It follows that only a halting computation can be correct and effective. Correctness presupposes halting.

An effective computation is one that successfully solves the problem it was designed to address. This means it can take any input from the problem's domain and produce a correct output in a finite amount of time. The computation halts for all inputs, indicating that the problem it solves has a definitive solution process. A problem is said to be decidable if there exists an algorithm that can determine the answer (yes or no) for any given input in a finite amount of time. Decidability implies the existence of an effective method for solving the problem. The connection between the two is that a decidable problem is one for which an effective computation exists. That is, for every instance of the problem, the corresponding computation will halt with a correct answer, showcasing that the problem can be algorithmically solved without exceptions.

An ineffective program refers to one that cannot guarantee a solution to the problem it addresses for all possible inputs. This might mean that the program does not halt for some inputs or that it cannot produce correct outputs for all inputs. An undecidable problem is one for which no algorithm can be designed that decides the answer (yes or no) for every possible input in a finite amount of time. For some inputs, it might be impossible to determine the answer algorithmically. The connection between the two concepts is that an undecidable problem is inherently linked to the concept of ineffective programs in that any program attempting to solve an undecidable problem will be ineffective by definition. This is because the program cannot produce a correct answer for all inputs in a finite amount of time, reflecting the undecidable nature of the problem.

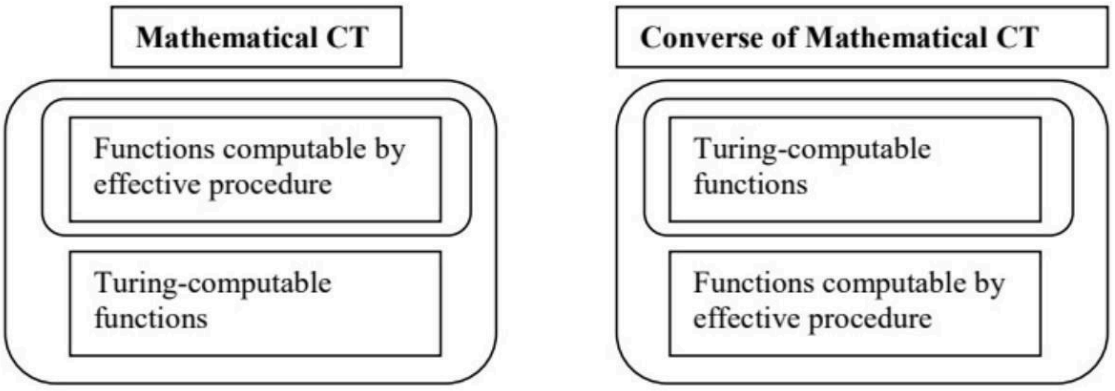
From these connections it follows that a non-halting computation does not solve any problem, therefore it's not effective.

2.15. Bi-conditional and implication forms of the CTT

The CTT can be stated in an implication form: *'Every effectively computable function can be computed by a Turing machine program P on any input i, which correctly halts within finite steps'.* The implication form allows for the possibility that there might be functions computed by a Turing machine that halts within finite steps but are not effectively computable. In other words, the set of functions computed by Turing machines could be a superset of the effectively computable functions. On the other hand, the biconditional form asserts that the set of effectively computable functions is exactly the same as the set of functions computed by Turing machines that halt within finite steps. This is a

stronger claim, as it rules out the possibility of any function being computed by a Turing machine that is not effectively computable.

In the definition of a computable process it was concluded that a computable function must be effectively calculable by definition because (Piccinini 2011)(Davis 1958, 11) the Turing machine must reach the final output with a terminal or halting state in a finite number of steps. It's effectively calculable because the process by which the computation is carried out is systematic, predictable, and can be completed without the need for infinite resources or time. In short, the calculation of the function is effective because the computation halts with a correct output. Because of this, in the context of the Church–Turing thesis, the biconditional form is generally accepted as the intended interpretation and therefore both the CTT and the converse of the CTT are considered as true. The following image shows a Venn-diagram of the CTT and the converse of the CTT (Piccinini, 2011).



The thesis aims to establish that the intuitive notion of effective computability is precisely captured by the formal concept of Turing computability (or equivalent formalisms like lambda calculus or recursive functions). However, it is important to note that the Church–Turing thesis is not a formally provable statement, but rather a hypothesis based on the equivalence of various computational models and the lack of counterexamples. The choice between the implication and biconditional forms in this context is more of a matter of interpretation and the intended strength of the claim being made. The implication form will be more important when the physical variant of the CTT is considered.

2.16. No computational method to evaluate effective computations

To connect the halting problem to the Church–Turing thesis and to highlight one key problem that the undecidability implies, we can turn to a proof by Martin Davis (Davis 1958, xvi-xvii) which shows that there is no meta-algorithm which can validate the correct or effective algorithms and processes. That is, it's not possible to evaluate computationally whether an arbitrary algorithm is correct, effective or – if we assume that only correct programs are meaningful – that if an algorithm is even meaningful. The proof by Davis is actually a variant of the proof of the undecidability of the halting problem. Here's a detailed explanation of the proof:

Initially, we suppose that an effectively calculable function is one for which there exists a definite, mechanical procedure (an algorithm) that allows for the computation of the function's value for any input from its domain (in this case, positive integers). We can then imagine all possible algorithms that compute such functions listed in an order determined by the length and alphabetical ordering of their

description in the English language. Each algorithm in this list is designated E_i, where i is its position in the list. The function associated in this way with E_i will be called $f_i(x)$.

The proof then revolves around demonstrating that a specific function $g(x) = f_x(x) + 1$ is not effectively calculable, thereby illuminating a fundamental aspect of algorithmic theory and its limitations. The function $g(x)$ is defined such that for any positive integer x, $g(x) = f_x(x) + 1$, where $f_x(x)$ is the output of the x^{th} algorithm when applied to x. Then the proof of non-effectiveness of $g(x)$ goes as follows.

1. First, we assume, for the sake of contradiction, that $g(x)$ is effectively calculable and that there exists an algorithm in the list E_i, which computes $g(x)$.
2. Then, according to the definition of $g(x)$, applying E_k to k would yield $g(k) = f_k(k) + 1$.
3. However, if E_k is supposed to compute $g(x)$, then applying E_k to k should give $g(k)$, which, by the construction of $g(x)$, is equal to $f_k(k)$ + 1.
4. This is a contradiction because $g(k)$ cannot equal $f_k(k)$ and $f_k(k)$ + 1 simultaneously.

The contradiction is caused by our initial assumption that $g(x)$ is effectively calculable, or more precisely that there is a list of all effectively calculable algorithms E_i and that $g(x)$ can be calculated by some specific algorithm E_k, which is on the list. The contradiction implies that we can conclude that $g(x)$ cannot be effectively calculable because it contradicts the premise that some algorithm E_k could compute it. This follows from the way $g(x)$ is constructed to always be one more than what any algorithm E_i in the list could compute for its corresponding input x.

The proof shows that by construction, $g(x)$ systematically evades calculation by any algorithm in the list E_i of all possibly effective algorithms. This result is profound, because it showcases an explicit example of a function that, despite being well-defined mathematically, cannot be computed by any mechanical process or algorithm. Davis then describes an attempt to construct an algorithm for computing $g(x)$ and explores why such an attempt is inherently flawed, ultimately proving that $g(x)$ cannot be computed by any algorithm. The initial attempt to develop an algorithm for $g(x) = f_x(x) + 1$ proceeds as follows:

1. Start with an input: Given an input x_0, the goal is to compute $g(x_0)$.
2. Generate the list E_i: Begin generating the list of all possible algorithms E_i, ordered as previously described, until you find E_{x0}, which is the x_0^{th} algorithm in the list.
3. Apply E_{x0} to x_0: Use E_{x0} to compute the function value $f_{x0}(x_0)$.
4. Compute $g(x_0)$: Add 1 to the result from the previous step to obtain $g(x_0)$.

The attempt is straightforward but it fails because of a fundamental issue with step 2, the generation of the list E_i. There are two critical problems with it. First, the assumption that it is possible to mechanically generate the list of all algorithms that compute effectively calculable functions is flawed. While it might seem feasible to list all possible strings of English language (including instructions for computations) in some order, the process described for generating E_i presumes it's mechanically possible to distinguish which of these strings actually represent valid algorithms. Second, the crucial problem arises in assuming that there exists a mechanical method to verify whether any given string of English text (presumed to represent computational instructions) indeed constitutes a valid algorithm for computing a function $f(x)$. This step is inherently non-mechanical because it involves interpreting the semantics of the instructions, understanding their validity as a computation method, and ensuring they correctly compute a function for all inputs in their domain.

This means that there is no meta-algorithm for validating effective algorithms, which implies that the process described for constructing an algorithm for $g(x)$ cannot succeed because it can't overcome the problem of algorithm verification. Davis then concludes that the failure lies in the assumption that one can mechanically determine whether an alleged set of instructions actually represents a valid algorithm for computing a function. This realization leads to a deeper conclusion: there is no algorithm or any computational method capable of deciding whether a given set of instructions constitutes a valid or effective algorithm for computing a function. **In short, this proof implies that the effectiveness or correctness of algorithms can't be computationally decided.**

2.17. The Church–Turing thesis as a quasi-empirical universal conjecture

There is a way in which the Church–Turing thesis and the concept of effective computation can be linked. The Church–Turing thesis, which is a philosophical conjecture rather than a formal statement, posits that any function that can be effectively computed can be computed by a Turing machine. While the thesis does not directly address the halting problem, it implies that for a function to be considered effectively computable, it must halt and produce a result in a finite number of steps. This aligns with the strict notion that effectively calculable functions are those that can be computed within a finite amount of time (Davis 1958, 11), ensuring that the computation is not just theoretically possible but practically realizable. In that case, it's possible to understand it as a universal conjecture that roughly states that all effective computations must halt correctly.

As an implication form of the Church–Turing thesis, we can state that *'all effectively computable functions must be computable by a Turing machine which halt and produce a result in a finite number of steps'*. Formulated this way, the Church–Turing thesis becomes a universal conjecture. As a universal conjecture, this form of the Church–Turing thesis implies that any function that we can compute (in the sense of an effective procedure) should be computable by a Turing machine. This makes it a broad, overarching hypothesis about the limits of computation. While the thesis is not provable in a formal mathematical sense (since it's about the relationship between informal and formal notions), it can be viewed as a quasi-empirical, universal conjecture. This means that, in practice, every new computation or algorithm discovered provides additional evidence supporting the thesis, as long as it can be shown to be computable by a Turing machine.

All universal conjectures have corresponding decision problems. A universal conjecture is a statement that asserts something is true for all elements in a particular set or domain. For example, Goldbach's Conjecture, *'all even numbers n greater than 2 can be expressed as the sum of two primes'* is a universal conjecture. When we consider Goldbach conjecture, we have to note that it has an implicit corresponding decision problem *'is even number n the sum of two primes?'* While universal conjectures make a general claim about a set of instances (e.g., all even numbers), the corresponding decision problem asks about a specific instance within that set (e.g., a particular even number n). Any potential future proof Goldbach's conjecture **presupposes** that the corresponding decision problem is decidable, otherwise there would exist some number n for which we would not be able to prove the conjecture. If there's any instance where we cannot determine whether an even number n is the sum of two primes, the conjecture cannot be fully proven because there's a gap in our knowledge regarding that instance.

Suppose we have the Church–Turing thesis as a universal conjecture: *'All effectively computable functions must be computable by a Turing machine which halts and produces a result in a finite number of steps.'* With any such universal conjecture – whether it is a mathematical conjecture like

Goldbach's conjecture or an empirical conjecture ─ it seems reasonable to ask whether it can be proven. Since the decidability of the corresponding decision problem is a prerequisite for the provability of the universal conjecture, it is natural to ask whether this decision problem is decidable.

In this context, the decision problem can be stated as follows:

For any function f and input i, is it decidable whether $f(i)$ is computable by a Turing machine that halts and produces a result in a finite number of steps?

This decision problem relates to the following conditions:

If a function f is effectively computable, then there exists a Turing machine P that, on input i, computes $f(i)$ and halts in a finite number of steps.

If a function f is not effectively computable, then no such Turing machine P exists that can compute $f(i)$ and halt in a finite number of steps.

If we believe that this universal conjecture should be provable for all instances, then theoretically, it should be decidable for all effective computer programs. This seems plausible since effective computability is often understood as what humans could achieve with paper and pencil. Moreover, the purpose of computers is to automate and check vast numbers of computations. However, for this to be possible, there would have to exist a computation H that evaluates whether a Turing machine P halts correctly for a given input i as follows:

If $P(i)$ correctly halts within finite steps,	then $H(\text{‘}P\text{’}, i) = 1$.
If $P(i)$ doesn't halt correctly halts within finite steps,	then $H(\text{‘}P\text{’}, i) = 0$.

This implies the undecidability of the halting problem right away. The halting problem tells us that there is no general algorithm to determine whether any given Turing machine will halt on a specific input, which complicates the provability of the Church─Turing thesis as a formal conjecture.

2.18. The physical Church─Turing thesis

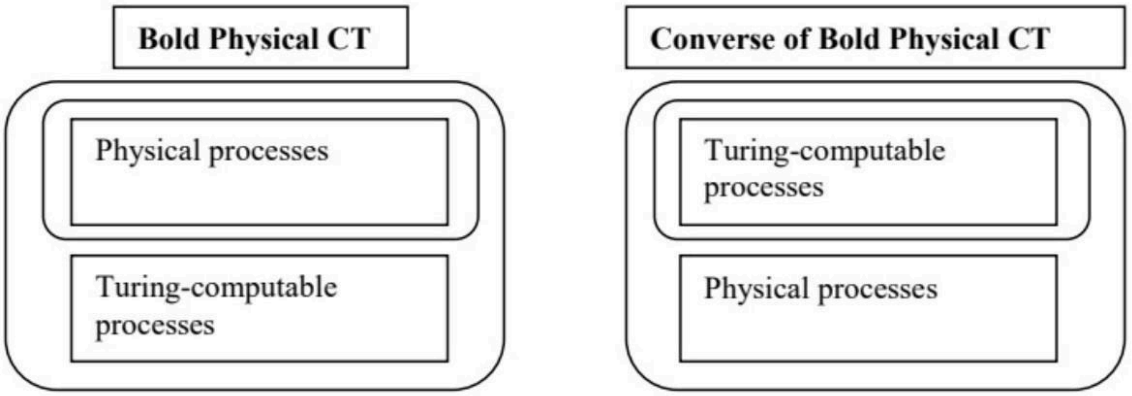
The Physical Church─Turing thesis (PCTT) is an extension of the Church─Turing thesis into the physical realm, stating that every physically realizable system can be perfectly simulated by a universal Turing machine operating by finite means. The PCTT is a stronger version of the Church─Turing thesis. Actually, the PCTT is considered as nothing but empirical and scientific conjecture at its core. As every scientific statement, the PCTT excludes the possibility of some states of the world (Popper 1959, 115). In this case, it asserts that there are no physical hypercomputers meaning that there are no physical computers which could for example compute or decide the halting problem. (Piccinini 2011, 754-755).

Refuting the Physical Church─Turing thesis (PCTT) would require demonstrating a physical process or constructing a physical *'computer'* that can perform computations beyond what is possible for Turing machines, showing that the class of physically computable processes is not a subclass of all Turing-computable processes. To refute the PCTT, we would need to provide concrete evidence of a physical computation that clearly falls outside the realm of Turing computability.

The literature of the physical Church–Turing thesis recognizes two versions: **strong and modest versions** (Piccinini 2011, 746-752). According to the strong PCTT, every physical process is computable by a Turing machine, and according to the modest PCTT, every function that is computable by a physical computer is also computable by a Turing machine. In short, the modest version only asserts that all physical computers can be simulated by a Turing machine, while the strong version asserts that all anything physical can be simulated by a Turing machine. More specifically, the implication form of the strong PCTT can be stated as: '*If a physical system is finitely realizable, then it can be simulated by a Turing machine'.* In contrast to this, the implication form of the modest PCTT can be stated as '*If a function f is computable by any physically realizable process (within the laws of physics), then f is computable by a Turing machine*'.

The **modest version** of the PCTT asserts that any function that can be physically computed can also be computed by a Turing machine. This does not necessarily imply that the Turing machine can simulate any physical system, but rather that the outputs of a physical *computation* can be reproduced by a Turing machine. In other words, this version holds that anything *computable* by any "physical device" or anything that a physical computer produces is also computable by a Turing machine. The strong version of the PCTT goes further by asserting that all physically real processes can be simulated efficiently by a Turing machine. While the existence of non-computable, true random processes in the physical world doesn't invalidate the modest PCTT - because those are not computational processes in the first place - , it certainly does invalidate the strong version (Piccinini 2011, 746-752). Because the modest version deals with only physical computational systems and computational systems in general, then when considering the ramifications of the halting problem, we only need to accept the modest version.

The PCTT implies that physical processes can be simulated computationally to any desired accuracy, allowing arguments about the physical world to be based on computation. This thesis is an extension of the Church–Turing thesis, which was originally proposed in a purely mathematical context, positing that all effectively calculable functions are computable by a Turing machine. The PCTT has implications for both theoretical physics and computer science, as it suggests that the fundamental processes of the physical world are, in principle, computable. This means that, given sufficient resources, it is possible to simulate any physical system through computational means, provided the system can be adequately described by physical laws. It implies that physical processes, from the motion of planets to quantum phenomena, can be simulated to any desired level of accuracy with a computational model, assuming the process is computable and we understand the underlying physics well enough. This allows us to say that the processes of nature can be modeled via a physical computer up to arbitrary accuracy. The PCTT wouldn't hold in a Newtonian universe where continuous and infinitely fine physical properties of objects are considered. Both Deutsch (Deutsch 1985) and Lloyd (Lloyd 2012) argue that current physical laws of quantum mechanics allow for such simulatability, because of quantization of physical properties. The following image shows a Venn-diagram of the PCTT and the converse of the PCTT.



It should be noted that compared to the CTT, the converse of PCTT does not hold. If the size of the universe is finite, then there are abstract Turing machine programs whose size is bigger than what can be realized in the physical universe.

„Consider Turing machines, with their tape of unbounded length. If the universe contains only a finite amount of matter-energy, then tapes of unbounded length might not be physically constructible. Furthermore, it seems unlikely that a finite user would ever be able to harness an infinite amount of matter-energy. All that we can construct is, most likely, finite approximations of Turing machines, such as our ordinary digital computers.... Arguments along these lines may be found in the literature on physical computability. They miss the way in which Turing machines are relevant to CT. All that CT says is that any function that is intuitively computable is computable by some Turing machine. Computability by Turing machines is the upper bound on what CT deems intuitively computable. And acting as an upper bound does not require being physically constructible." (Piccinini 2011, 28)

This implies that for the PCTT, only the implication form of the statement is correct, according to which *'every physical process can be perfectly simulated by some computation'*.

2.19. The physical Church–Turing thesis and physicalism

The PCTT on its own – by stating that all physically realizable computations are also realizable by a Turing machine – does not make any specific claims about the physical nature of all reality. It focuses only on the relationship between physical computations and Turing-computable functions. This means that accepting the PCTT does not inherently mean a commitment to physicalism. It leaves open the possibility that there could be computational processes that are not physically realizable, or aspects of reality that are non-computational and yet physical.

The PCTT – combined with physicalism – suggests a universe where all processes, including mental phenomena, are computable. This underpins computational theories of mind, which propose that mental states and processes are the outputs of computational processes in the brain. Physicalism posits that everything is physical or at least supervenes on the physical (Stoljar 2024). This encompasses all phenomena, including mental and computational processes. The PCTT suggests a similar universality from a computational perspective, by stating that any physical process can in principle be simulated by a Turing machine or a computational model that adheres to the principles of classical computation.

If physicalism holds true then all events, including computation are rooted in physical processes.This implies that any computation that can be carried out must also be achievable through means, which is in line with the core idea of the PCTT. From a physicalist perspective the PCTT could be viewed as an extension because it mirrors the physical foundation of all computable functions.

Assuming both the PCTT and physicalism means all physical processes are computable and also that there exist no computational or non-computational processes which are not physical. The acceptance of both principles supports a reductionist approach in science, where complex phenomena can be understood in terms of simpler ones. This does not necessarily mean that all scientific explanations will become purely computational or that they will ignore emergent properties, but it suggests that at a fundamental level, everything operates according to computable physical laws. This viewpoint may appear deterministic as it suggests that if all operations are computable theoretically with sufficient data we could predict the future states of any physical system.

A fundamental idea of physicalism is that physical laws and theories can provide complete explanations for all phenomena. If certain phenomena **cannot be captured** by computable functions, this would suggest that our current understanding of physical laws is incomplete or that there are limits to how fully such laws can explain the complexities of the universe. This would challenge physicalism's claim of explanatory completeness. If the Physical Church─Turing thesis (PCTT) were found not to hold, indicating that there are processes in the physical world (and by extension, potentially processes involved in knowledge and cognition) that cannot be fully simulated or captured by mechanistic computational models, it would suggest limitations to the mechanistic approach in representing and processing all forms of knowledge. It would imply that there are aspects of knowledge or cognition that elude the currently known mechanistic model of representation, which is Turing-computational.

Neither the PCTT nor physicalism logically implies the other. The PCTT is a claim about the computational nature of physical processes, while physicalism asserts that everything is ultimately physical. The PCTT aligns well with physicalism because if all physical processes are computable, it suggests a universe that is fundamentally describable by physical laws and computations. If we hold the PCTT but not physicalism, it allows for the possibility of non-physical processes that are not captured by the PCTT, such as dualism or idealism in the philosophy of mind. If we hold physicalism but not the PCTT, it suggests that there might be physical processes that are not computable, challenging the notion that the universe is fundamentally computational. However, from a pragmatic point of view, usually both the PCTT and physicalism are held together.

That is because if there were physical processes that cannot be simulated by Turing machines or any computational model, this would imply the existence of aspects of the physical world that elude a core principle associated with some interpretations of physicalism ─ that all physical phenomena can be understood, in principle, through physical laws that are computable or mechanistically describable. For empiricists who are also physicalists, the mechanisms through which we acquire and process knowledge must be physical in nature, since empiricism relies on the physical interaction between our sensory organs and the world. If we hold both physicalism and empiricism as true but acknowledge but reject the PCTT, the implication is that there must be some other mechanistic model capable of explaining how knowledge is processed and acquired through physical means. If certain aspects of knowledge acquisition or processing fall outside the realm of Turing computability, a new model or paradigm might be necessary to explain these processes in a way that aligns with physicalism. Since this would be quite an esoteric position without actually showing this *'other'*, new type of mechanistic process, it follows that from a pragmatic point of view, a physicalist should hold on to the PCTT. The converse is also true if we're empiricists. By considering the PCTT as an empirical hypothesis, it would be contradictory to hold on the one hand that each physical process is computational in nature

and yet hold on the other hand also as an empirical hypothesis that there are computational processes which are non-physical.

If both the PCTT and physicalism are considered true, it implies that there can be no non-computational proofs, as the nature of the universe is seen as fundamentally computational.

2.20. The physical Church–Turing thesis as an empirical thesis implies the halting problem

Similar to the original CTT, the principle of empiricism has implications for the PCTT. Empiricism holds that all knowledge and therefore a knowledge embodied by a universal conjecture comes from verification or falsification, through direct observation or experimentation. Since the PCTT is considered as nothing but empirical conjecture at its core, it follows that we have to consider the PCTT in the following modest (Piccinini 2011, 752) form as a scientific hypothesis: *'A physical computer B can simulate the computations of computer A under any given corresponding inputs (i), by halting and producing results that correctly correspond to what A would compute.'* More formally, this can be expressed as: $\forall(i)(A(i)correctly\ halts \supset B(i)\ correctly\ halts)$. Again, the principle of empiricism holds that all knowledge and therefore a knowledge embodied by a universal conjecture comes from verification or falsification, through direct observation or experimentation. In an objective reality facts exist independently of our perceptions and can be verified through appropriate means. But this means that this conjecture then has a corresponding empirical decision problem, which can be stated in the following form:

If B(i) halts correctly within finite steps, then it is determinable by (either verifiable or falsifiable empirically) that B(i) halts correctly.

If B(i) doesn't halt correctly within finite steps, then it is determinable by (either verifiable or falsifiable empirically) that B(i) does not halt correctly.

But since with the PCTT we also hold that all physical processes can be simulated by a computation, then certainly empirical verification ─ which is also a physical process ─ should be a process which can be carried out by a physical computation. So there has to exist some physical computation H which does the job of empirical halting evaluation in the following way:

If $B(i)$ correctly halts within finite steps, then $H(\text{‘}B\text{’}, i) = 1.$
If $B(i)$ doesn't halt correctly halts within finite steps, then $H(\text{‘}B\text{’}, i) = 0.$

Which implies the undecidability of the halting problem. It is the joint result of the PCTT and the halting problem that there is no effective way to determine the correctness of computational simulations. So the halting problem arises if we want to establish by some form of computation if the Physical Church–Turing thesis is true or false.

2.21. Problems with the physical Church–Turing thesis

The PCTT's treatment of finiteness is quite vague, as it can be interpreted as having an infinite scope while asserting finiteness. The assertion that *„**any** finitely realizable physical system can be perfectly simulated by a universal computer operating by finite means" (Deutsch 1985)* brings to light a

paradoxical interpretation of finiteness within the PCTT. By stating *'any'* when talking about the *'finitely realizable system'*, the PCTT can imply a scope that encompasses an infinitely wide array of systems, as long as each can be realized in a finite manner. This creates a contradiction. The claim begins with a premise of finiteness but then extends to potentially infinite scenarios under the umbrella of *'*any system*'. This renders the PCTT'*s interpretation of finiteness as overly broad and not truly reflective of physical constraints.

Another core problem with the PCTT is that every computation in the physical world is not just an abstract process but a real process, which is subject to the laws of physics (Ferry and Porod 2023). This should mean that the halting status of a physical process is measurable, with each measurement applying some finite, measurable quantity. However, this finite nature also implies limitations ─ energy, time, precision. In the physical world, finiteness isn't just about being able to enumerate steps or components; it's also about the practical constraints that govern how and to what extent computations can be realized. The PCTT on its own doesn't state anything about these constraints.

The PCTT's claim of perfect simulation is problematic from an empirical point of view, as perfect simulation is impossible in the physical world due to inherent approximations and limitations. Physical systems are subject to noise, quantum uncertainties, thermal fluctuations, and countless other variables that make perfect replication impossible. So when the PCTT posits that any physical system can be perfectly simulated, it overlooks the fundamental nature of physical reality, where every observation or measurement introduces some degree of error. In practice, simulations are tools for approximation, prediction, and understanding, rather than flawless replication. This distinction is critical because it highlights the gap between the theoretical computability of a system and its practical, physical realizability. From this it follows that the expectation of perfect simulation is much more aligned with an idealized, Platonic view of the universe. A universe where abstract computational constructs can seamlessly mirror the complex, nuanced reality of physical phenomena.

2.22. Measurement limitations of the PCTT

The PCTT does not provide a clear framework for addressing the precision of measurements required for physical computation. In real world scenarios every physical computation must contend with the constraints of measurement including sensor precision, noise interference and uncertainties inherent in quantum systems. These factors introduce approximations and limitations that go beyond the abstract concept of computation as defined in the PCTT. While the PCTT suggests that any physical system can be simulated by a Turing machine, it does not provide a clear framework for addressing the practical limitations and measurement requirements which are necessary for the simulations to be meaningful and correct.

For the PCTT to be an empirical hypothesis, it must incorporate an empirically measurable halting condition. Without the ability to halt and produce results, the thesis lacks practical verifiability, as computations need to produce finite, observable outcomes that can be empirically tested. Consider again the modest form of the PCTT: *'A physical computer B can simulate the computations of computer B under any given corresponding inputs (i), by halting and producing results that correctly correspond to what A would compute.'* Without a measurable, empirical halting condition, the thesis simply restates or reformulates the concept of the universal Turing machine. Without such a halting condition, correctness can't be checked or verified, meaning that without such a halting condition, the PCTT is not a correct empirical hypothesis.

The inclusion of the halting condition in the consideration of the PCTT could mean that it is actually an empirical hypothesis about the nature of physical reality and computation. It suggests that for a

physical process to be computable — and for a physical computer to be considered correct in its computation —, there must be a finite, observable result.

The original formulation of the halting problem and also Turing's proof of its undecidability doesn't operate with any constraint on the measurability of the non-halting condition. In fact, there is no empirical measurement whatsoever involved in the original formulation of the halting problem. While the PCTT sounds physical, it's not enough to consider any computation to be subject to physical laws, for example the laws of thermodynamics (Ferry and Porod 2023, 7) or that in physical reality each computational process either halts or doesn't halt. By simply considering the PCTT in connection with arguments regarding the halting problem, we're still giving infinite resources to the supposed halting program.

In order to correct these mistakes, usability constraints on the Physical Church–Turing thesis (Piccinini 2015, 249–256) can be outlined to ensure that a physical process counts as computable in a way that is relevant to both computer scientists and physicists. These constraints have to be specified to make the notion of usability precise for a finite observer. Inputs and outputs of a computation must be readable by a finite observer (Piccinini 2015, 251). This means they can be measured as inputs or interpreted as outputs to any desired degree of approximation. A physical process must be executable (Piccinini 2015, 249) if a finite observer can initiate it to generate the values of a desired function until a readable result is produced. This includes the ability of the observer to understand which function is being computed, to read the process's inputs and outputs, and to construct the system exhibiting the process. The process should run without requiring intuition, ingenuity, invention, or guesses from the observer (Piccinini 2015, 252). The process should not need to be redesigned or modified for different inputs. It should work uniformly across various inputs without the need for adjustment. The process should generate results that are correct at least some of the time. Reliability (Piccinini 2015, 253) includes the generation of correct outcomes whenever the process does produce results. These constraints — especially the first two — ensure that a physical process is practically computable and usable by finite observers, including humans or any other beings of similar cognitive capacities.

2.23. Naive application of the halting problem to the PCTT

A naive application of halting problem to the PCTT overlooks all measurement constraints. In physical reality, computations are constrained by time and energy. The assumption that computations can run indefinitely without addressing the limitations of physical resources (like energy availability or the finite nature of materials) simplifies the complex nature of real-world computations. In reality, no physical computational process such as the halting program H can run forever due to these limitations. Some problems may require an impractical amount of time to solve, making the instant decision implied by the construction of H unrealistic. Another aspect of a naive application is the assumption that all properties of physical processes can be precisely captured by computational models without the inherent measurement uncertainties and the need for these properties to be quantifiable in finite terms. It fails to consider that in the physical world, any property that a computation seeks to simulate or predict must be measurable within a finite amount of time and with finite precision. If a physical process is a computational process, this doesn't imply that all abstract computational processes that can be imagined to be carried out by abstract Turing machines are also physical and real processes.

Prima facie physicalist, but in this sense naive applications of the halting problem are used in the discussions around free will (Lloyd 2012). Scientific determinism holds that given the perfect knowledge of a physical law and the initial conditions, the future results or the future events the follow from the initial conditions can be predicted with perfect accuracy — where a prediction is nothing but a

type of computation which regards some future physical state of affairs. Then an imaginary superhuman, but still physical predictor ─ called Laplace's Demon ─ should be able to predict all future events with perfect accuracy. If we hold a form of the PCTT as true in this situation, then by applying the result of Turing's proof we can conclude that even the superhuman Laplace's Demon wouldn't be able to predict at least some future physical state of affairs. This shows that there is something wrong with the physicalist-empiricist applicability of the halting problem.

Combined with the proof of the undecidability of the halting problem, the PCTT tells us about the limits of what we can compute about the physical world. It implies that if a physical process cannot be computed by a Turing machine, it challenges our understanding of computation or the process itself. With the help of the assumption of the physical Church–Turing thesis and the assumption of physicalism that only physical entities exist, it can be ensured that any physical decision process can be simulated on a Turing machine. And if it can be simulated on a Turing machine, then the physical systems under analysis can only perform calculations that can be recursively calculated with Turing machines. Thus, all limitations characteristic of Turing machines will also apply to physical systems. In this way, if a certain Turing machine computation instantiating the Halting Function (Sipser 2006, 173-182), cannot be computed, then there cannot exist any physical computer system that can calculate it.

Consider the following argument, which leads to a philosophical contradiction. According to Turing's proof and the diagonal argument, it turns out that no halting evaluator H could determine whether $D('D')$ halts or not, so the output value of $H(‘D’,’D’)$ can't be produced by any computational process. Following a naive application of the PCTT, any physical process must be a computational process (Deutsch 1985)(Lloyd 2012). Therefore we conclude that the value of $H(‘D’,’D’)$ can't be produced by any physical process either, and also that H can't be a real physical process itself. Therefore we conclude that no physical validator H could determine whether $D(‘D’)$ halts or not.

Following the PCTT we can also conclude that being a real physical process presupposes that the program halts after a finite amount of steps and since it can't be determined by any physical process whether $D(‘D’)$ halts or not, we conclude that no physical validator H could determine whether $D(‘D’)$ is physically realizable or not.

This would lead to a form of agnosticism regarding the empirical knowability of the physical realizability of $D(‘D’)$ and also contradicts the common sense physicalist view that $D(‘D’)$ can be determined to be non-realizable from a physical point of view.

This problematic argument implies another problem for physicalism. Suppose Turing's proof is true. Then although $D(‘D’)$ being physically realizable or not can't be computed, yet we (physical humans) can somehow see according to the common sense physicalist view that $D(‘D’)$ can be determined to be non physically realizable. If $D(‘D’)$ is evaluated as non-physical by humans, then we physical humans know more than computers. So our human ability to evaluate the undecidability of certain problems seems to go beyond what the PCTT would predict, suggesting that human reasoning might have capabilities that exceed those of computational systems (Lucas 1961)(Hofstadter 1979)(Penrose 1989).

In order to see how and why this contradiction arises and if it can be resolved in a way that both the physicalist, computationalist and empiricist stances can be considered without contradiction, the naive application of halting problem has to be analyzed thoroughly. But even before that, in a physicalist framework, we have to analyze if it is possible if computation can be understood as a purely physical process which does not rely on anything abstract.

2.24. Computation in physical systems

Computation is the process of performing mathematical calculations or logical operations systematically to obtain a desired result. This process can be considered as abstract, which exists purely in mathematical theories and models, or it can be considered as concrete which is exclusively physical, like computers and biological brains.

The Turing machine ─ as presented in the former chapter ─ is a theoretical construct. It is usually described as not a physical machine but an *'abstract'* one which is designed to represent any computation through a set of simple, universal operations. Its theoretical nature allows it to represent any algorithm, regardless of the specifics of the physical hardware on which that algorithm might run. Models of computation, like the Turing machine, are not tied to any specific hardware or physical implementation. This *'abstraction'* allows computer scientists to study computational processes independently of the physical characteristics of machines. It means that the same algorithm can run on a desktop computer, a server, or a smartphone, despite the vast differences in their physical construction and operation. This however doesn't imply that a Turing machine exists in some abstract, Platonic realm, even if this is what's very often argued for.

Consider the following argument, according to which the *'standard of computation'* is a universal, non-physical property upon which the individual working methods of the physical instances of computers are based.

„The property of a brain mechanism or a computer mechanism which makes it work according to the standards of logic is not a purely physical property, although I am very ready to admit that it is in some sense connected with, or based upon, physical properties. For two computers may physically differ as much as you like, yet they may both operate according to the same standards of logic. And vice versa; they may differ physically as little as you may specify, yet this difference may be so amplified that the one may operate according to the standards of logic, but not the other. This seems to show that the standards of logic are not physical properties. (The same holds, incidentally, for practically all relevant properties of a computer qua computer.)" (Popper 1972,

In computational terms, *'sameness'* refers to the multiple realizability (Bickle 2020) of different computational models (e.g., Turing machines, lambda calculus, Post machines) to simulate each other and perform the same computations, despite differences in their physical implementations or theoretical underpinnings. That is because the Church─Turing thesis supports the abstraction of computation by showing the equivalence between various models of computation. Before Turing, Alonzo Church (Church 1932, 1936) developed the lambda calculus, another formal system capable of expressing any computable function. The Church─Turing thesis posits that what can be computed by the Turing machine can also be computed by the lambda calculus and vice versa, along with other models like Post systems and Markov algorithms. This equivalence of solutions is what suggests that the essence of computation lies not in the specific details of these models but in the abstract notion of computability itself. So it is the ability to maintain computational equivalence through various mediums which implies the idea of universality. It suggests that the essential properties that define a system's computational capabilities are not tied to its physical attributes. Therefore, the universality of computation is not limited by physical constraints but is defined by the abstract *'logical'* operations a system can perform. Actually, the empirical understanding of PCTT ─ compared to the abstract nature of the CTT ─ implies that such sameness of computations is not a priori true. So it might be that one day it turns out that some form of computation is not the *'same'* as the other.

There is another sense however, in which *'sameness'* can be understood. Computers seem very stable and reliable objects and this suggests that any computation performed by them is also beyond

usual physical imperfections. This can't be any further from the truth — it's easy just to think of any failure in hardware embedded computers — such as a failure of the control module of an elevator — to see that it is not true. What gives the impression of reliability is that the fundamental operation of transistors, including those used in NAND gates (Mano and Ciletti 2017, 90), relies on the use of high and low voltage levels to represent binary information (1s and 0s). This binary operation, based on high (HI) and low (LO) action potentials or voltage levels (Boylestad and Nashelsky 2019, 7), is a core principle of reliable digital electronics. Also, a computer can be seens as consisting of a series of functionally complete NAND gates or chips. By applying just a single XNOR gate to two NAND gates, the result is a verifier circuit. That is because the XNOR gate is a digital logic gate that outputs true or logic high (1) only when its two inputs are equal. In other words, it outputs a 1 when both inputs are 0 or when both inputs are 1. This setup can be seen as a verifier in the sense that the XNOR gate (Mano and Ciletti 2017, 92) is verifying whether the outputs of the two NAND gates are in agreement (either both have processed their inputs to produce a logic high or both have produced a logic low). A typical computer can utilize many such physical forms verification which make it seem unusually stable and reliable. However, this is not a result of any abstract capacity, but simply a result of the physics applied. Actually, such forms of physical verification can be applied to different physical computers which can be understood as a kind of abstraction process in the form that the resulting, physically verified computation is less and less dependent on the particular imperfections of the singular elements. With this explanation, it's possible to argue that an abstraction does not lead out of the physical world. It is a physical process itself. In general, abstraction involves moving from a detailed description of a physical system to a less detailed one while maintaining the essence of the computational process. Even if this process is not based on a series of electronic chips, it is guided by physical theories that describe the common features or the common behavior of the system in question. Even physical versions of the Turing machine have been built (https://aturingmachine.com/). In what follows, when the Turing machine is discussed or when the expression 'a physical Turing machine' is used, it will be understood as using this particular solution.

There is another question, which is frequently asked in the philosophical literature of computation: what differentiates physical systems that compute from those that do not (Piccinini and Maley 2021)? Consider as a first answer to the question of differentiability the simple mapping account, which is a concrete — physicalist — account of computation (Putnam 1960). It attempts to describe how abstract computational descriptions (e.g., algorithms or Turing machine specifications) can be related to physical systems (like digital computers or potentially other physical objects) in a concrete manner. This account operates on the principle that a physical system can be considered as performing a specific computation if there's a mapping between the states and transitions described by a computational model and the physical states and transitions of the system. Under the simple mapping account, a physical system S performs a computation defined by a computational description C if there is a mapping from the states ascribed to S by a physical description to the states defined by C, such that the state transitions between the physical states mirror the state transitions between the computational states.

This account is *'simple'* because it focuses on the possibility of such a mapping, without imposing strict requirements on the nature of the states or the causality of transitions beyond their alignment with a computational model. However, this simplicity also leads to criticisms, particularly because of its liberal attribution of computational activity to physical systems, potentially leading to an overly inclusive or trivial understanding of what it means to compute. The problem is that under this account nearly any physical system could be seen as performing any computation with the right mapping. Consider a rock (Chalmers 1996). By carefully selecting parameters and interpretations, we could devise a mapping from the states and transitions of a computational model to the physical states and changes a rock undergoes over time, for example temperature changes. If such a mapping is enough to classify the rock as a computing device — for example to *'compute'* the temperature of the Sun —, then the definition of computation becomes so broad that it loses its usefulness and explanatory

power. The argument implies that calling a rock a computer would trivialize the notion of computation, because it would mean that any physical system could be considered computational.

There have been several responses to refute the implications of the *'can a rock compute?'* argument. One such response is that computation requires the existence of counterfactual conditions (Block 1978). This means a physical system does not implement a computation through a static mapping of its current states but must also be capable of undergoing state transitions that correspond to the computation under various other conditions, not just the ones it actually experiences. Consider a digital clock designed to display the current time by advancing its displayed numbers every second, accurately reflecting the passage of time.The clock's program moves between states representing each consecutive moment in time (for example from 1:59 PM to 2:00 PM). By considering the counterfactual alternative, if the clock were in a different state, for example displaying 1:58 PM, and one second passes, it would then transition to displaying 1:59 PM. This shows that if the clock were in a different state, then it would also correctly map the passage of time. This is not true of the rock, which would not *'perform'* the same actions, for example it would not compute the temperature changes of the Sun at night.

Another response regards the causal structure of the system (Chalmers 1995, 2011). According to this view, for a system to compute, its state transitions must not only map onto those of a computational model but also be causally related to each other in a way that mirrors the causal structure implied by the computational model. It can also be argued that computation involves an element of design (Piccinini and Maley 2021), meaning that a system computes not just because of the abstract mapping of its states but because it was designed to perform computations. According to the argument which uses semantic content (Fodor 1975) of computations as a differentiating property, for a system to be genuinely computing, it must manipulate symbols or states in a way that is meaningful. While all such accounts successfully differentiate between computational and non-computational physical processes, from the point of this investigation, both the counterfactual account and the semantic account are argued for extensively in the following chapters. A physical computation is only a meaningful computation if it correctly represents a family of states of affairs in the physical world such that this representation or mapping is also counterfactually complete.

3. The physical Church–Turing thesis and the halting problem

Seth Lloyd – in his article The Turing Test for Free Will (Lloyd 2012) – draws philosophical conclusions about the predictability of physical systems from the well known uncomputability of the halting problem (Turing 1936). The argument in question dealing with the unpredictability of *prima facie* deterministic physical systems has its own history looking back more than 70 years from Karl Popper (Popper 1950) to D.M. MacKay (MacKay 1967) to Adolf Grünbaum (Grünbaum 1971).

Given a variant of the physical Church–Turing thesis (abbreviated as PCTT) (Deutsch, 1985) (Piccinini, 2011) it is supposed that the uncomputability of the halting problem is true for any physical system. If a prediction is a specific type of physical computational task (Lloyd 2012, 3602), where to predict the decision of a physical computational system – or computer – is to compute the output of any such computer or to compute that the computer halts or doesn't halt on any output, then from the proof of the uncomputability of the halting problem it can be shown that the physical computational agent won't be able to carry out some predictions. If we take this argument as true, then even by considering the physical system in question as deterministic, it still won't be able to carry out some predictions. This implies that the resulting indeterminacy of these decisions in question arises whether or not the physical system computing the decision – the physical computer – is deterministic or not.

„The theory of computation implies that, even when our decisions arise from a completely deterministic decision-making process, the outcomes of that process can be intrinsically unpredictable, even to – especially to – ourselves." (Lloyd, 2012, p. 3597)

Lloyd's argument doesn't propose to fully explain free will (Lloyd 2012, 3600–3601), it only presents a necessary condition for it in a physicalist setting. It asserts that even if the physical universe is initially considered as deterministic, it still follows from logical arguments that some future decisions of physical agents which regard some physical states of affairs are unpredictable or undeterminable. If the freedom of will can be at least partially explained by this unpredictability then it can also be said that all physical systems characterized by the PCTT are free because of the uncomputability of the halting problem. From the above chain of arguments we can only investigate a specific part, which seems to be the core of the argument. Is the uncomputability of the halting problem actually true or physically meaningful if we also consider the physical Church–Turing thesis as true? Can they be both true at the same time?

The key thesis of this current doctoral analysis is to show that the core argument under investigation, which asserts that there exist physical decisions whose outcome can't be predicted, is false.

3.1. Key concepts used

As presented in the former chapter, the **Physical Church–Turing thesis** (PCTT) asserts the following (Deutsch, 1985): *„Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means"*, or just simply: every finite physical system can be simulated by a (universal) Turing machine. The PCTT is an empirical statement about the capabilities of physical systems. In his argument, Lloyd does not reference the PCTT directly, but his two assumptions imply the thesis: *„In order to address the physics of free will with mathematical precision, we have to make some assumptions. The first assumption that we make is that our deciders are physical systems whose decision-making process is governed by the laws of physics. The second assumption is that the laws of physics can be expressed and simulated using Turing machines or digital computers (potentially augmented with a probabilistic 'guessing' module in order*

to capture purely stochastic events). The known laws of physics have this feature. These two assumptions imply that the decision-making process can be simulated in principle on a Turing machine." (Lloyd 2012, 3601)

Physicalism is the philosophical position that everything in the universe, including mental phenomena like consciousness, can be fully explained by physical matter and energy, governed by the laws of physics. Physicalism combined with the PCTT implies that there are no computations which are not physical computations.

In traditional **Platonism** — originating from the ideas of the Plato — abstract forms or ideas are considered to be the most real and fundamental constituents of reality. These forms are timeless, unchanging, and exist independently of human thought and physical reality. When applied to formal systems and computations, Platonism is the belief that mathematical objects — such as numbers, functions, sets —, and also logical truths, proofs, and the principles of computation exist in an abstract realm, independent of human discovery (Linnebo 2024).

Another key concept used in Lloyd's argument is the halting problem. As presented in the former chapter, we can say that a computer program — called the halting program H — solves the halting problem if it can look at any other program and tell if that will run forever or not (Crossley et. al 1972, 38-39). Suppose that there exists a halting program H that can calculate the following: $H(\text{'}P\text{'}, i) = P(i)$, when P stops on input i, and $H(\text{'}P\text{'}, i) = 0$ when P doesn't stop on input i. **Turing's proof of the undecidability of the halting problem** shows that no such halting program H exists (Turing 1936). In Lloyd's argument, the formal notation is a bit different. Instead of H, f is used. Instead of P, d is used and instead of i, k is used.

3.2. The argument from the uncomputability of the halting problem

In Lloyd's argument, a variant of the PCTT is assumed, by which every physical process is a computational process. Then, according to the argument (Lloyd 2012, 3602) a prediction f about a computational decision $d(k)$ is itself a type of a computational task where to predict a computation d is to calculate all possible outputs on all possible inputs k for a given future computation d — if that computation halts. However, if $d(k)$ doesn't halt then the prediction must assign it a specific $'doesn't\ halt'$ or $'fails'$ output value.

„Can anyone, including the deciders themselves, predict the results of their decisions beforehand, including whether or not a decision will be made? A simple extension of the halting problem shows that the answer to this question is 'No'. In particular, consider the function $f(d, k) = d(k)$, *when* T_d *halts on input* k, $f(d, k) = F\ (for\ Fail)$ *when* T_d *fails to halt on input* k. *Turing's proof of the uncomputability of the halting function can be simply extended to prove that* $f(d, k)$ *is uncomputable. So the question of whether a decider will make a decision at all, and if so, what decision she will make, is in general uncomputable."* (Lloyd 2012, 3602)

The way Lloyd defines a prediction requires the predicting machine to incorporate the capabilities of both a Universal Turing machine — defined in the former chapter — and a halting program. If the computer d to be predicted halts on the input k then the predicting machine should calculate the same $f(d, k)$. Following Turing's diagonal proof, the halting program can't compute such values, since its output value must be either $d(k)$ or $Fail$. This task requires a universal Turing machine (UTM),

because in this variant of the halting problem, if $d(k)$ halts on some output, then the program f at $f(d,k)$ computes the same as $d(k)$ would compute. However, if the computer to be predicted doesn't halt on an input then the prediction must assign a specific *'doesn't halt" or 'fails'* output value to the task. A UTM is not able to do that (Crossley et. al 1990, 37–41), only a halting program can assign such output values. The argument is supposed to show that even in a deterministic universe there exists a type of unpredictability for physical agents which could explain why physical agents feel free. The starting point of the argument is the supposition built on the PCTT, that we are considering deterministic and nothing but physical deciders that are no more complex than Turing machines.

In Lloyd's argument, the formal notation is different from the canonical variant. In what follows, instead of f, H will be used. Instead of d, P will be used and instead of k, i will be used.

3.3. Decisions and predictions

Based on Lloyd's arguments, both decisions and predictions are considered as computational processes because of the physical Church–Turing thesis. Lloyd argues that the decision-making process inherently involves computation, especially when considered from the perspective of computational complexity and the theory of computation. **Decisions**, according to Lloyd, are such physical processes which are outcomes of complex computational processes that are influenced by a combination of deterministic and probabilistic factors. Lloyd extends the discussion to **predictions**. The prediction of one's own decisions is also a physical computational process. The act of predicting outcomes, including the outcomes of decision-making processes, is presented as a computationally intensive task that, in many cases, requires as much computational effort as making the decision itself.

In Lloyd's usage of the terms, following PCTT, a physical **decision** and a **prediction** of a decision are computational tasks. If a prediction is a physical computational task, then how and why does a prediction differ from other computational tasks? How does it differ from a mere decision? Based on the PCTT, we can suppose that **decisions** are computational processes that involve evaluating options and outcomes based on a set of inputs and internal reasoning algorithms. Then there are two main distinctions. First, **predictions**, especially about decisions, are higher-order computational processes. They not only involve the direct computation required to simulate or anticipate outcomes but also encompass meta-level computations that assess future computational decision processes. Second, predictions are about future decisions. The reason for this distinction is that for computations or decisions regarding events in the past, the concept of halting can be straightforwardly applied. If a computation or a decision has been carried out to process or analyze past events, then by the time of evaluation, it has either halted or not. In practice, past computations are evaluated based on their completion and correctness relative to known outcomes or data. Therefore, the halting status of such computations is not subject to the same undecidability as future oriented, predictive computations.

We can connect the concept of a decision to the former, formal definition of a decision problem UP. A decision problem UP involves determining specific properties or answers for all instances in a given domain UD. For example, if UP is about *'determining primality'*, UD would include all natural numbers, and the question Q would be whether a number is prime. The problem seeks a procedure to consistently and correctly answer these questions. A problem is decidable if there exists a Turing machine T that can, for every instance in UD, correctly answer in finite steps, with 1 for *'YES'* and 0 for *'NO.'* Therefore, the decision problem's solution relies on finding a computational method to

address a yes/no question for any instance within UD, highlighting the problem's computability and decidability.

Based on the definition of a decision problem, a **decision** is the outcome of a computational process that resolves a specific question from a class of questions Q, related to a particular domain UD, by determining whether a certain binary property or criterion applies to an instance within that domain. The decision is binary in nature, meaning it can either affirm YES or negate NO the presence of the property in question for the given instance. Also, a decision is an instance of the application of a decision-making process that is algorithmically determined. A decision problem is considered decidable if there exists a total Turing machine that can make these decisions (outputting 1 or 0) for every possible input in a finite amount of time. The process that leads to a decision is uniform across all instances in the class of questions Q. This means that the same algorithmic process or Turing machine is used to make decisions for any question derived from the domain UD.

In Lloyd's discussion within the context of the physical Church–Turing thesis and decision-making processes, the use of the term 'decision' can be linked to the formal definition of a decision, yet with a broader philosophical and physical interpretation. In Lloyd's framework, a decision is the outcome of a computational process conducted by a physical system (which could be a human brain, a computer, or any decision-making entity). Similar to the formal definition, decisions in Lloyd's context have binary outcomes (yes or no), corresponding to the resolution of a specific question or problem. Lloyd extends the concept of decisions beyond abstract computation to include all physical processes, adhering to a variant of the physical Church–Turing thesis. This suggests that decisions, in the broad sense, are not only outcomes of abstract computational models but also of real-world physical processes that can be simulated computationally. When can we say that such a **physical** computational decision is true or if it has a true **physical** representation? In order to answer this, we have to turn to some definition of representation and truth for physical formal systems. In this case, we can apply a definition or representation by (E.Szabó 2017, 8–9) for the physical formal system L:

„...one is entitled to say that a formula A represents or means a state of affairs a in U, if the following two conditions are met:

(a) There exists a family $\{A_\lambda\}_\lambda$ of formulas in L and a family $\{a_\lambda\}_\lambda$ of states of affairs in U, such that $A = A_{\lambda 0}$ and $a = a_{\lambda 0}$ for some λ_0 .

(b) For all λ,

if a_λ is the case in U, then $\Sigma_L \vdash A_\lambda$
if a_λ is not the case in U, hen $\Sigma_L \vdash \neg A_\lambda$"

Based on this definition, a decision as a physical computational process can be formally described as follows:

(a) There exists a class of questions of state of affairs $\{a_\lambda\}_\lambda$ in the physical universe to which a general computational decision P with a class of input-output pairs $\{i_\lambda, o_\lambda\}_\lambda$ corresponds.

(b) Then a true, meaningful or effective decision P is such that for all pairs $\{a_\lambda, i_\lambda\}_\lambda$

if a state of affairs a_λ is the case, then $P(i_\lambda) = 1$,
if a state of affairs a_λ is not the case, then $P(i_\lambda) = 0$.

Based on this, we can give a short definition of a decision as follows. **A decision is the outcome or result of a physical and also computational process which is a conclusive resolution of a question about some physical state of affairs.**

A **prediction** regarding a decision outcome is a meta-decision about the occurrence of a decision-making event within a computational process. It involves determining whether a computational process will reach a definitive decision (yes or no outcome) on a given problem or question. This type of prediction assesses the future state of a decision-making process, specifically whether it will successfully conclude with a decision or not. A prediction is a higher-order decision, a meta-decision on decision-making, focusing on the binary outcome of another decision process. It involves reasoning about the future state of a computational decision-making process. Unlike broader predictions that may concern various future properties or events, this prediction specifically targets the outcome of a decision-making process. The primary interest is whether the process culminates in a decision. Making such a prediction may involve simulating the decision-making process in question or employing computational methods to estimate the likelihood of the process reaching a conclusion.

Given a decision problem characterized by a domain UD and a class of questions Q, **a prediction** regarding a decision outcome is a meta-level decision about whether a computational process tasked with resolving a question from Q will produce a binary decision within a specified timeframe or under given conditions. Therefore, a prediction H as a computational process can be formally described as follows:

(a) Suppose there exists a class of questions of computational decisions $\{P_\lambda, i_\lambda\}_\lambda$ in the physical universe to which a computational prediction H with a class of $\{'P'_\lambda, i_\lambda\}_\lambda$ corresponds, where $'P'_\lambda$ represents the λ-th Gödel-number of decision P in an enumeration of computational decisions and i_λ represents the λ-th input to that computational decision.

(b) Then a correct, true, meaningful or effective prediction H is such that for all such pairs,

if a future decision P_λ on input i_λ produces an outcome (halts), then $H('P'_\lambda, i_\lambda) = 1$,
if a future decision P_λ on input i_λ does not produce an outcome (halts), then $H('P'_\lambda, i_\lambda) = 0$.

Based on this, we can give a short definition of a prediction as follows. **A prediction is the outcome or result of a physical and also computational meta-level decision process which is a conclusive resolution of a question about whether some physical computational decision process will produce a decision outcome or not.** From this definition it is already visible that a prediction is a meta-decision process, but it is also visible that H is nothing but the halting program, because it needs to evaluate whether any future decision process halts or not. From this, based on Turing's proof of the undecidability of the halting problem, we can arrive at Lloyd's conclusion: there exist some future decisions which can't be predicted. But if this is true, then we can further specify which future decisions which can't be predicted. Based on Turing's proof of the undecidability of the halting problem, we can define a future anti-diagonal decision process D, decision process which can't be predicted:

(a) Suppose there exists a class of questions of computational decisions $\{H_\lambda, 'i'_\lambda, i_\lambda\}_\lambda$ in the physical universe to which a computational decision D with a class of $\{i_\lambda\}_\lambda$ corresponds, where $'i'_\lambda$ represents the λ-th Gödel-number of computational decision I in an enumeration of computational decisions and i_λ represents the λ-th input to that computational decision.

(b) Then a decision D is such that for all such pairs

if $H_i('i'_i, 'i'_i) = 1$, then $D(i_i)$ doesn't produce an outcome (doesn't halt),
If $H_i('i'_i, 'i'_i) = 0$, then $D(i_i)$ does produce an outcome (halts).

This means that based on Turing's proof outlined above, there exist some allegedly self-referential future decisions which can't be predicted.

3.4. Bounded and unbounded predictions

We can categorize predictive computations further. The first type is an **unbounded prediction**, in which there are no constraints ─ for example no time-constraints ─ for the original computations $P(i)$ to be predicted by H and there are also no such constraints for H either. It can be possible that $H(P,i)$ runs for a very long, even indefinite amount of time before H actually finishes its computation regarding $P(i)$. In a **bounded prediction**, this is not allowed, as the prediction should definitely halt for example within some given time-frame. The idea behind allowing for indefinitely running unbounded predictive computations is that it is quite possible that after some very long time some seemingly non-halting computation actually finishes and halts. This is problematic however from an empirical point of view, as in the real world we can't allow predictions to run forever. However, in Lloyd's argument, the unbounded version is used for a general argument and similar unbounded versions are used in Laplace-demon type of arguments (Popper 1950) against scientific determinism, where it is imagined that a powerful predictor demon H_D can run even for an indefinite amount of time and yet it still won't be able to finish some predictive computations. Since according to Lloyd's argument, there are bounded and unbounded physical, computational predictions, we can analyze the ramifications for each.

3.5. The core argument ─ Argument A

With all the above in mind, the argument for unpredictability can be summarized the following way:

a. Suppose that the physical Church─Turing thesis (PCTT) is true.
b. Then given the 1. PCTT and 2. any physical system, the halting problem is uncomputable by that system.
c. Since the halting problem is uncomputable, it's possible to draw philosophical consequences of epistemic limitations for any physical system, for example that some physical decisions can't be predicted.

The focus on this current investigation is on the second part (b.).

The core argument ─ we can call it Argument A ─ can be outlined as follows:

1. A variant of the Physical Church─Turing thesis is assumed to hold. According to this, all physical processes can be simulated by Turing machine computations.
2. According to Turing's proof and the diagonal argument used in the proof, no halting evaluator H can determine whether a self-referential computation such as $D('D')$ halts or not.

3. The output value of $H('D','D')$ can't be produced by any computational process H.
4. Following a naive application of PCTT, the physical process of a halting evaluation ─ or in Lloyd's term a prediction of a decision ─ carried out by a physical agent H process is a computational process which disregards all real physical constraints. In such a naive application of the PCTT, there can also exist self-referential physical computational processes $D('D')$.
5. Therefore by following the application of PCTT, it follows that the output value of $H('D','D')$ can't be produced by any physical process H.
6. Therefore we can also conclude that no physical validator H could determine whether $D('D')$ halts or not.

Argument A is about an unbounded predictive computation, in which there are no time-constraints for the original computations $P(i)$ *or* $d(k)$ to be predicted by H *or* f and there is also no time-constraint for H *or* f either. It can be possible that $H('P', i)$ *or* $f(d,k)$ runs for a very long, even indefinite amount of time before H *or* f actually finishes its computation regarding. Since the same type of unbounded and allegedly physical predictions are used in Laplace-demon type of arguments against scientific determinism, it has to be analyzed before analyzing the bounded variant if the argument which uses unbounded predictions holds.

If Argument A holds, then there are philosophical consequences, both negative and positive. **On the negative side**, there exist some future decisions about physical states of affairs which can't be predicted. A further implication of this is that these predictions can't be evaluated for correctness. Therefore some computational models of the physical world can't be evaluated for correctness or meaningfulness. **On the positive side**, it's possible to link this fundamental unpredictability to philosophical notions which can be seen as arising from some sort of unpredictability, such as the freedom of will or qualia. These philosophical concepts are notoriously hard to explain in a purely physicalist setting and the usage of the halting problem combined with the PCTT gives a way to explain these.

3.6. Problems with Argument A

Predictive computations about the future ─ especially those which involve complex or self-referential computational systems ─ fall into the category of computations whose halting status ─ and by extension, correctness regarding future outcomes ─ cannot always be determined by physical computational agents. Once time passes, future computations will become past computations and once these become past computations, their halting behavior and correctness can be evaluated. However, according to the argument, **there exist some future decisions about physical states of affairs which can't be predicted**, so for some future oriented computations the predictions won't even be doable or executable. They simply wouldn't finish until the future predicted event happens. So basically the agent doing the prediction won't be able to execute the prediction before the event it predicts comes to pass.

3.7. The challenge of evaluating meaningfulness without predictions

In traditional decision-making evaluations, the correctness of a decision is often assessed against a prediction or an expected outcome. If a prediction process cannot conclude with a definitive prediction due to undecidability, then there's no predictive benchmark against which to evaluate the decision's

correctness post hoc. Without a prior prediction, assessing whether the decision process aligned with what *'would have been predicted'* becomes speculative. This situation introduces a degree of indeterminacy, as the evaluative process lacks a clear standard or expected outcome to compare against the actual decision outcome. In the absence of a concrete prediction, any post-hoc evaluation risks falling into post-hoc rationalization, where the observed outcome is retroactively justified without an original predictive benchmark. This can undermine the objective evaluation of the decision process's effectiveness or correctness. So the worry is that if Lloyd's argument is correct then that for some decision P it's not possible to evaluate whether P as a general computation really corresponds to the class of questions about the states of affairs. Specifically for some self-referential computational decisions, it's not possible to decide whether they are correct or not. Lloyd argues that for certain complex or self-referential decisions, **not only is predicting their outcomes challenging, but so too is evaluating their correctness post hoc**, due to the inherent computational or logical complexities involved. In the context of Lloyd's argument, this means there may exist decision processes for which 1. predicting their outcomes is impossible due to computational limits or self-referential complexities and 2. determining post hoc whether the decision process correctly addressed its intended questions or accurately represented the states of affairs is also problematic, given the undecidability or paradoxical nature of the process.

Can any fact of the physical universe be predicted and meaningfully represented by physical agents*?* If a decision can't be predicted, then it can't be evaluated for correctness or meaningfulness. Any decision must be based on some theory, in particular it must be based on some computation. The meaningfulness of the decision and the computations was defined above. This means that the meaningfulness of a decision regards the question whether D as a general computation really corresponds to the class of questions about the states of affairs. Now if some decision D can't be predicted on a class of inputs, then this means that it can't be evaluated D on that class of inputs really corresponds to its intended class of questions about the states of affairs. Can any fact of the physical universe ─ including the allegedly physical fact of the halting status of computation $D(\text{`}D\text{'})$ ─ be predicted and meaningfully represented by some physical computation H*?* If we hold on to the principle of empiricism, then the answer to the question can only be yes. However, if Argument A is true, then it contradicts empiricism, because it implies that there is a fact of the world ─ $D(\text{`}D\text{'})$'s halting behavior ─ which can't be evaluated for correctness by any physical agent. According to the argument, **if a decision can't be predicted, then it can't be evaluated for meaningfulness.** Therefore argument A expresses a kind of agnosticism towards physical and empirical knowledge. Empiricism relies on the idea that observation and experimentation can uncover any truth about the physical world. Argument A suggests that there is a boundary to this.

If Argument A holds, it doesn't just contradict empiricism in the straightforward sense of 'there are things we can't know through observation.' It introduces a class of phenomena within the deterministic physical universe that are ─ due to their computational nature ─ beyond the reach of empirical predictions. For some future oriented computations the predictions won't even be executable, because they simply wouldn't finish until the future predicted physical event happens. This is especially problematic for contemporary AI systems (e.g. Matuszka, Czuczor, Sóstai 2018) whose knowledge representation, evaluation and self-learning skills are based upon predictability and evaluability. This contradicts the empiricist and physicalist premises that systematic observation can uncover all facts about the physical universe. **This contradiction with empiricism suggests that Argument A might be false. Therefore, the key thesis of this current analysis is to show that the core argument under investigation, which asserts that there exist physical decisions whose outcome can't be predicted ─ Argument A ─ is false.**

3.8. Additional philosophical questions

Accepting Argument A as valid requires some philosophical presuppositions. One such presupposition is the existence of a physical computer which realizes a universal Turing machine. Such a machine can be considered to exist in reality only if the tape of the machine is infinitely extensible or if we require the machine to run for an indefinite amount of time (Crossley et. al. 1990, 40). Another philosophical presupposition is the converse of the physical Church–Turing thesis (Piccinini 2011, 734). Since the undecidability of the halting problem requires that the halting function operates over all Turing machines (Crossley et. al. 1972, 37–39), it is also required that all finite computers must be realizable. If the number of elementary particles in the known physical universe is finite it is questionable that these presuppositions can be correct.

We can ask a number of other philosophical questions regarding the argument, such as: Is the freedom of will reducible to general unpredictability? Can we really suppose that the brain realizes a universal Turing machine and not simply a non-universal one? This current analysis takes a charitable position regarding these questions. The point is only to show that even if we consider these philosophical consequences of the argument as non-disruptive, we can still show that Lloyd's core argument runs into contradictions.

3.9. Is it meaningful that *'no program H can compute the halting problem'?*

In Lloyd's argument, the formal notation differs from the usual one, but it can be translated without loss of meaning. In the following sections, the more canonical notation will be used with H instead of f, P instead of d, and i instead of k. A decision will be translated as a computation regarding a decision problem, a prediction will be treated as a special type of computation of a decision problem which regards some future physical state of affairs. Consider the following (meta-) metatheoretical statement, which follows from Turing's proof: *'The physical computer H can't compute the halting problem.'* We have to consider that the PCTT constrains the computational capacity of any physical agent – or any physical prediction decider – to some Turing machine M at most. Since Lloyd's argument uses Turing's proof, if we take physicalism seriously, we have to assume that Turing's proof itself is physically meaningful in the sense that it is obtainable or computable by physical agents such as humans or other physical and computational agents such as a computer M. Knowing this, can the seemingly meaningful (meta-) metatheoretical statement be calculated by any physical agent M?

To analyze the problem, we have to first ask: when can we say that such a metatheoretical statement is true or if it has a true representation? In order to answer this, we can turn to the definition of representation and truth for physical formal systems (E. Szabó 2017, 8-9) applied above. Following this definition, a meaningful representation requires the *representing* computation to correctly halt on each instance of the problem in question. Utilizing this definition, for a physical computer program H to represent the variant of the halting problem *'does computer I halt on input 'I'?'* this would mean the following. Let $I(`I') = 0 \text{ or } 1$ represent the metatheoretical fact that the program $H$ which simulates the $I$-th program in some standard enumeration halts on the $`I'$-th input. Each such $I(`I') = 0 \text{ or } 1$ metatheoretical fact would be represented by an $H(`I', `I')$ output value if the following condition holds:

For each $(I, 'I')$ number pairs it is true that,

if $I('I') = 0$ *or* 1 is the case, then $H('I', 'I') = 1$
if $I('I') = 0$ *or* 1 is not the case, then $H('I', 'I') = 0$

Can any fact of the physical universe ─ including the allegedly physical fact of the halting status of a computation $D('D')$ defined in the following section ─ be meaningfully represented by some physical computation H? Based on Turing's proof of the uncomputability of the halting problem (Turing 1936) (Crossley et. al. 1972, 37-39), no such computation can be realized by H. That is because we can define the following anti-diagonal program D. D's program is special, because it uses the computations of $H('I','I')$ as a subroutine (Rayo 2019, 329). This means that what it does is the following sequence of actions in sequential order. First, D takes one input i, then computes $H('I','I')$ using i on both argumentum values of H. Second, $H('I', 'I')$ returns an output value (0 or 1). Then D uses that output value in the following way:

If $H('I', 'I') = 1$, then $D(i)$ doesn't halt.
If $H('I', 'I') = 0$, then $D(i)$ halts.

Program D on input $'D'$ by definition contradicts the output of $H('I', 'I')$:

If $H('D','D') = 1$, then $D('D')$ doesn't halt.
If $H('D','D') = 0$, then $D('D')$ halts.

This results in the following contradiction(s):

If $D('D')$ halts, then $H('D','D') = 1$.
If $H('D','D') = 1$, then $D('D')$ doesn't halt.

If $D('D')$ doesn't halt, then $H('D','D') = 0$.
If $H('D','D') = 0$, then $D('D')$ halts.

If H is uncomputable, then what Turing's proof entails is that the allegedly representing computation H doesn't halt correctly on each instance of the problem in question. If we follow the definition of meaning presented above, then we can say that H is **not meaningful** or that H can't represent the halting problem. This is also the case for D too, and that is because it uses H's computations as a subroutine, meaning that in order to compute $D(i)$, it would have to wait until $H('I', 'I')$ finishes its computation and halts. This computation however never gets finished, so if H is non-computable, then D is non-computable and thus non-meaningful and physically unrealizable either.

A similar conclusion can be reached if instead of the definition of representation and truth for physical formal systems by E. Szabó (E. Szabó 2017, 8-9), we would simply look at what the physical Church–Turing thesis entails. The PCTT tells us about the limits of what we can compute about the physical world. The physical systems under the assumption of the PCTT can only perform calculations that can be calculated with Turing machines. Thus, all limitations characteristic of Turing machines will also apply to physical systems. In this way, if a certain Turing machine computation cannot be computed, then there cannot exist any physical computer system that realizes it. The upshot is again, that according to Turing's proof and the diagonal argument, no halting evaluator H

could determine whether $D('D')$ halts or not, so the output value of $H(`D', `D')$ can't be produced by any computational process. Therefore we can conclude that the value of $H(`D', `D')$ can't be produced by any physical process either. Following the PCTT we can conclude that H and D can't be real physical computational processes.

We could almost stop at this point in analyzing Lloyd's argument, because if we take physicalism seriously, then this entails that what Lloyd calls physical predictors - the physically instantiated H and D agents - are not physically real entities and therefore their predictions can't be real predictions either. This would of course mean that Lloyd's argument is false in light of these results.

However, we have to be careful with such conclusions. The above argument against Lloyd's views assumes that Turing's proof is true. But this also means that if we take physicalism seriously, we would have to assume that Turing's proof itself is physically meaningful in the sense that it is obtainable or computable by physical agents such as humans or other physical agents such as a physical computer M. The following argument ─ Argument B ─ investigates if this supposition itself leads to contradictions.

To analyze the problem further, we have to first ask whether *'the halting problem is uncomputable'* itself is a meaningful statement. Therefore, representing the meta-metatheoretical statement of the truth of uncomputability of the halting problem by M can be the next step in the argument. Let $H(`I', `I') = 0 \text{ or } 1$ represent the meta-metatheoretical fact that the halting program evaluating the I-th Turing machine on the $`I'$-th input halts on $0 \text{ or } 1$. The meta-metatheoretical fact would be represented by an $M(`I', `I')$ output value if the following condition holds:

For each $(`I', `I')$ number pairs it is true that,

if $H(`I', `I') = 0 \text{ or } 1$ is the case, then $M(`I', `I') = 1$
if $H(`I', `I') = 0 \text{ or } 1$ is not the case, then $M(`I', `I') = 0$

In this case, M represents a computational solution to the decision problem *'does computer H halt on input $`I'$ ─ where $`I'$ is a variable representing both the Gödel-number of any physical computer and the number of its input?'* . Suppose that this $M(`I', `I')$ machine exists. By its definition, M would be a physical agent ─ or simply a program ─ which can compute the output of **any** meta-metatheoretical decision problem and therefore also decide whether any process such as H itself halts. Then, if the undecidability of the halting problem is true in the sense of being computable or decidable by M, as a special case of the above truth condition, $M(`M', `M')$ would be able to also decide the meta-metatheoretical problem that M in fact doesn't halt on $`M'$. This means that the output value of $M(`M', `M')$ would be $0$. However, the output value of $M(`M', `M')$ halting on $0$ directly contradicts that $M(`M')$ halting on 0 or 1 is not the case:

if $M(`M') = 0 \text{ or } 1$ is the case, then $M(`M', `M') = 1$
if $M(`M') = 0 \text{ or } 1$ is not the case, then $M(`M', `M') = 0$

Therefore M can't compute or represent the meta-metatheoretical statement either, that *'H can't calculate the H halting program'*. If the criterion of the truth of a statement is that the above condition holds, then it is impossible that this meta-metatheoretical statement can be true in the sense that it could be calculated by M (Sóstai 2023).

This argument ─ which we can call Argument B ─ proposes that M is a machine capable of deciding any meta-metatheoretical decision problem, which would include determining whether a given program I halts on input $'I'$. It shows that the specific case where M attempts to determine its own halting status with M as the input leads to a contradiction: the inability of M to accurately determine its own halting status when analyzing itself $(M('M', 'M'))$ contradicts the broader claim of M according to which it can represent the halting status of any Turing machine (including $H('I', 'I')$).

This shows two key consequences. First, we have to be cautious before considering Turing's proof as being obtainable and then usable by a physical agent which can decide the halting status and therefore the physical meaningfulness or physical realizability of the agents, what Lloyd calls physical predictors. Second, Since Lloyd's argument ─ Argument A ─ uses Turing's proof then if we take physicalism seriously, this would also have to assume that Turing's proof itself is physically meaningful in the sense that it is obtainable or computable by physical agents such as M. But according to the above argument, this runs into a contradiction, showing that given the assumption of physicalism, Lloyd's argument is false.

3.10. Further problems with Argument A

Argument B presented above does not show a direct contradiction, where for example evaluating $H('I', 'I')$ as non-halting would itself be a contradictory act for M. To show that this would lead to a direct contradiction, we can extend Argument B if it is allowed that H can also use the computations of M as a subroutine. In order to show that this leads to direct contradiction, we have to assume again that Turing's diagonal proof of the undecidability of the halting problem is correct and meaningful in the sense that it can be known by physical agents such as M. This assumption will then lead to a contradiction.

Turing's original proof doesn't go as far as to state that $H('D', 'D')$ doesn't halt. The proof only shows that computing either $H('D', 'D') = 0$ *or* $H('D', 'D') = 1$ is impossible, because it leads to a logical contradiction. When a physical agent M evaluates $H('D', 'D')$ as meaningless or ineffective, then M has to perform a semantic evaluation of $H('D', 'D')$. Why would this imply that $H('D','D')$ doesn't halt?

That's because if $H('D','D')$ were a meaningful computation, then by definition any such computation would halt. We can suppose that $H('D','D')$ is syntactically valid, so at least it's possible to run the computation because it is sufficiently defined. So it is supposed that $H('D','D')$ describes some actual computational process. But then if $H('D','D')$ is effective, then by definition it halts. Which can't be the case if Turing's proof is correct. So if following Turing's proof, M concludes that either case of halting $(0 \text{ or } 1)$ is impossible for $H('D','D')$, then it means Turing's proof together with the supposition that $H('D','D')$ is a real computational process implies that $H('D','D')$ is ineffective or non-meaningful.

What does it mean that H is not meaningful? It specifically means that H on a specific input doesn't halt with the correct value. This can happen for two reasons, if we think that $H('D','D')$ is at least a syntactically valid computation. It can happen because H *on* $D('D')$ doesn't halt or it can happen because H halts on $D('D')$, but not with the correct value. In the first case, it is reasonable to argue

that H could use M as a subroutine, because what M computes has something to do with H's halting status. In the second case this would mean that there is some discrepancy between H's prediction about $D('D')$'s halting status and $D('D')$'s actual halting status. If this is the case, it is also reasonable that H could use M as a subroutine, because what M computes has again something to do with H's output, which has a direct relevance to $D('D')$'s halting status. However, in both cases, using M as a subroutine to H leads to the same halting paradoxes as before.

If Turing's proof is interpreted as $H('D','D')$ not halting and M could be used as a subroutine by H therefore would mean that $M('D','D') = 0$. If this computation of M can be used as a subroutine for H, then H would evaluate that $H('D','D')$ doesn't halt. If $H('D','D')$ doesn't halt then it follows that $D('D')$ doesn't halt. This follows from the special setup of the program D, which uses H's output as a subroutine. If $D('D')$ doesn't halt, then $H('D','D') = 0$. This follows from the halting program H's own definition. Then, if $H('D','D') = 0$, then $D('D')$ halts. This follows from D's program. But if $D('D')$ halts, then $H('D','D')$ = 1, which is a contradiction again. If Turing's proof is interpreted as $H('D','D')$ halting, but not on the correct value, then we could assume it would be possible to just flip its output value with a program N. We can define such N as follows:

If $H('I','I') = 1$, then $N('I','I')$ halts on 0
If $H('I','I') = 0$, then $N('I','I')$ halts on 1

Then the following holds for $N('N','N')$:

if $N('N','N')$ halts correctly, then $H('N','N') = 1$
if $N('N','N')$ doesn't halt correctly, then $H('N','N') = 0$

This again leads to a contradiction:

if $N('N','N')$ doesn't halt correctly, then $H('N','N') = 0$
If $H('N','N') = 0$, then $N('N','N')$ halts on 1

What these arguments show is that there can't exist any physical computational agent such as M in the sense that if M would be an agent which could semantically evaluate the halting program H as meaningless. That is because if M's computations could be used for calculating that $H('D','D')$ doesn't halt or for calculating that $H('D','D')$ halts but incorrectly, then this would lead to contradictions. But according to the PCTT, there was nothing special in the physical computer M which differentiated it from a physical human agent M'.

Argument A has a contradictory consequence which can be interpreted along platonist and physicalist positions. We have assumed both that the PCTT and also physicalism holds. Also, by considering physicalism as valid, have assumed that Turing's proof and the diagonal argument is meaningful and it is obtainable by physical agents. Prima facie, this implies that $D('D')$'s halting behavior can't be computed and that the output value of $H('D','D')$ can't be computed by any physical system.

But when considering Turing's proof as being obtainable by a physical agent, this leads to a contradiction because physical agents carrying out Turing's proof would be able to calculate that the output value of $H('D','D')$ can't be computed by any physical system. This would contradict Turing's

proof that $D(‘D’)$'s halting behavior can't be computed and also implies that the output value of $H('D','D')$ can be computed, which is a contradiction again.

Therefore, any physical agent would run into the same contradictions as M would. It follows that if we consider physicalism as true, then this contradiction can’t be resolved. Any physical agent such as M faces the same contradictions. How can these contradictions be resolved? The following sections deal with some potential solutions.

3.11. Is the proof of the uncomputability of the halting problem a platonic proof?

The PCTT doesn’t necessarily imply physicalism (Piccinini 2011). If the assumption of physicalism is discarded, then the contradiction seems to be easily avoidable. If for example Turing’s proof is true, but not in the form that it can be enumerated by a physical computer or a physical human agent, then of course the former proofs of contradictions don’t hold. This line of resolution is quite popular among different philosophers (Lucas 1961)(Hofstadter 1979)(Penrose 1989), but it places human cognition in the realm of the supernatural ─ which contradicts both physicalism and empiricism.

Can’t we just suppose that the proof of the uncomputability or the halting problem is a kind of platonic truth and that a *'knowledge'* of such truths is possible? The consequences of such a philosophical position are the following. First, even with the *'knowledge'* of such non-physical truths no exclusively physical decider would be able to know that it itself is unpredictable ─ this follows from the results of the former sections. Second, if no physical decider can know that, then only non-exclusively physical ─ or non Turing limited ─ deciders can know that.

If that’s true, the original argument would look like this:

a*. Suppose that the physical Church─Turing thesis is true, but there also exist non exclusively physical deciders.
b*. Then given the 1. PCTT and 2. any physical system, the halting problem is uncomputable for that system.
c*. Since the halting problem is uncomputable, it’s possible to draw philosophical consequences of epistemic limitations with the knowledge of non-exclusively physical *'deciders'* which regard any physical system.

If we’re supposing a non-purely physical ─ or non Turing-limited ─ decider then the whole argument doesn’t concern her and her alleged limitations since she doesn’t have any such limitations. What would be the point of such an argument for the decider? From the point of Lloyd’s argument, this line of thought also misses the premise that unpredictability is a consequence for physical agents. What this argument shows is that either we accept physicalism ─ and then assuming Turing’s proof and the PCTT leads to a contradiction, or else if we discard physicalism, then the whole point of the original argument ─ which is about physical deciders ─ is lost.

3.12. Can M_2 calculate that M_1 can't represent the halting program?

Consider the following argument:

a. The M_1 physical system can't represent the halting problem.
b. But some other M_2 physical system can compute that the M_1 physical system can't represent the halting problem.

In this interpretation, there would exist a hierarchy of physical computers (M_n, M_{n+1}, M_{n+2},..,M_{n+m}) which under the assumption of the PCTT, would all be realizing Universal Turing machines, and a computer M_{n+1} would be able to calculate that a computer M_n can't calculate the n-th halting problem. Let's suppose that this is true. Then take any undecidable problem, for example the word problem for groups (Novikov, 1955). Applying M_2 we could then ask if M_1 halts or not when M_1 simulates Turing machine W, which is the machine that tries to decide the word problem on (X, Y) word pairs. If $W(X, Y)$ halts, then M_1 also halts and M_2 calculates that M_1 halts. But if $W(X, Y)$ doesn't halt then M_1 doesn't halt, but at the same time M_2 can calculate that M_1 doesn't halt. But this means that M_2 can calculate that the word problem (X, Y) for groups can't be calculated and M_2 decided an undecidable problem. And this is not possible.

3.13. Can M guess that no program H can compute the halting problem?

In this hypothetical case, M does not compute the value of H, but M computes the output value of the much less general *conjectural* H or H_C for short, which is in fact a completely different Turing machine than H_C. In this hypothetical case, one could say that after a finite number of steps, the CH performing the conjectural computation must in any case produce an output that halts at 0 or 1. Of course, this *'solution'* has some consequences. First, it is clear from the above description and from the knowledge of the halting problem (Crossley et al. 1972, 37-39) that H_C gets the wrong solution most of the time. Second, consider the case in which the *'definite finite number of steps'* is extended to the *'indefinite number of steps'*. In this case, if the tape of the machine containing the program of the Turing machine to be evaluated can contain the description of any finite-length Turing machine, the problem is reduced to the universal problem, because the H_C should cover an arbitrary number of steps, i.e., it should then evaluate *any* finite-length Turing machine. But if the H_C evaluates *all* Turing machines, we are back to the original halting problem. (Crossley et al. 1972, 37-39).

3.14. Summary

Argument A assumes the PCTT, according to which all physical processes are equivalent to computational processes and can be simulated by a Turing machine. This assumption handles physical systems as computational systems capable of executing decision-making processes. Decisions are then defined as the outcomes of computational processes that resolve specific questions about states of affairs in the physical world. In physical systems viewed through the lens of PCTT, decision-making is equated to executing a computational algorithm that processes inputs to arrive at a conclusion. Predictions are extended decisions where the outcome of a computational

process or a future state of a physical system is forecasted. Predicting the outcome of a computational process, especially in physical systems, involves computing the process's future state based on its initial conditions and its program. Predictions are therefore higher-order computational tasks that determine the results of decision-making processes. Making predictions about computational processes directly invokes the halting problem, particularly when predictions aim to determine whether a decision process will halt or continue indefinitely. Since the halting problem is undecidable, this imposes a fundamental limit on the predictability of computational processes. By applying the halting problem to the predictions of physical systems' behaviors ─ as allowed by the PCTT ─, the argument demonstrates that there exists an inherent unpredictability in physical systems.

In an unbounded prediction, there are no constraints for the original computations $P(i)$ to be predicted by H. It can be possible that $H(P,i)$ runs for a very long, even indefinite amount of time before H actually finishes its computation regarding $P(i)$. In a bounded prediction, this is not allowed, as the prediction should definitely halt for example within some given time-frame. If Argument A ─ which utilizes unbounded prediction ─ is true, then it contradicts empiricism, because it implies that there is a fact of the world ─ $D(\text{'}D\text{'})$'s halting behavior ─ which can't be evaluated for correctness by any physical agent. Therefore Argument A expresses a kind of agnosticism towards physical and empirical knowledge.

If H is uncomputable, then what Turing's proof entails is that the allegedly representing computation H doesn't halt correctly on each instance of the problem in question. If we followed a physicalist definition of meaning (E. Szabó 2017, 8-9), then we could say that H is not meaningful or that H can't represent the halting problem. This would also be the case for D too, and that is because it uses H's computations as a subroutine, meaning that in order to compute $D('I')$, it would have to wait until $H(\text{'}I\text{'}, \text{'}I\text{'})$ finishes its computation and halts. This computation however never gets finished, so as an intermediate result, we could conclude that if H is non-computable, then D is non-computable and thus non-meaningful and physically unrealizable either.

However, if we take physicalism seriously, we would have to assume that Turing's proof itself is physically meaningful in the sense that it is obtainable or computable by physical agents such as humans or other physical agents such as a physical computer M. Only if Turing's proof itself is physically meaningful would we be able to assert the former intermediate result. The key argument presented against Argument A ─ Argument B ─ investigates if this supposition itself leads to contradictions.

Argument B can be summarized as follows. We assume the Church─Turing thesis. We assume physicalism. We also assume ─ following Turing's proof ─ that there exists a program H, which is not effectively or meaningfully computable. This leads to a contradiction, because one the one hand, this implies that a physical and therefore computational agent M could calculate that a computation $H(\text{'}D\text{'},\text{'}D\text{'})$ is not effectively or meaningfully computable. On the other hand, by following Turing's proof, either M itself could not meaningfully represent any meta-metatheoretical decision problem or if such computation by M would be obtainable by H, that would eventually contradict that $H(\text{'}D\text{'},\text{'}D\text{'})$ is not computable.

M is a hypothetical computational entity, that is is assumed to have the capability to compute or decide the outcomes of complex computational problems, including meta-metatheoretical problems. These are higher-order problems that involve making decisions about other computational processes, such as determining whether another computation will succeed or fail. The extension of Argument B allows for the possibility that H (the halting evaluator) can use the computations of M as a subroutine.

This introduces a new layer of complexity and leads to a direct contradiction. If H uses M as a subroutine, it introduces a feedback loop where H depends on M to make a determination about whether certain computations halt. This situation leads to a logical contradiction because of the nature of the halting problem and the constraints of Turing's proof. Suppose that Turing's proof is interpreted by M in a way that $H('D','D')$ does not halt. Now, if H uses M as a subroutine, and M computes $M('D','D') = 0$, then based on the construction of D and H and their dependencies on each others computations, this implies that M determines that $D('D')$ does not halt. Given that $M('D','D') = 0$, H would then evaluate that $H('D','D')$ does not halt. If $H('D','D')$ does not halt, it follows (according to the construction of the program D) that $D('D')$ does not halt. This is because D is designed to behave in a way that if $H('D','D')$ does not halt, $D('D')$ itself does not halt. However, based on the definition of the program D, if $D('D')$ does not halt, then by the setup, $H('D','D')$ should output 0. But if $H('D','D')$ outputs 0, then $D('D')$ is supposed to halt according to H, which contradicts the previous step where we concluded that $D('D')$ does not halt according to H. This shows that the process of using M as a subroutine to H leads to a point where the evaluation of $D('D')$ becomes inconsistent, resulting in a logical contradiction.

The upshot of both Argument B, as well as its extension presented against Argument A is that Turing machines are incapable to even calculate that they can't calculate the output value of any arbitrary program. In Lloyd's terminology regarding computational predictions, if the epistemic capacity of a physical decider is only Turing machine limited ─ meaning that there is no bound on the measurement of halting behavior according to the PCTT ─, then no physical machine can decide that it can't predict a decision. Contradicting Lloyd (Lloyd 2012, 3597), we can say that if humans are physical beings then humans can't even know that they are intrinsically unpredictable to themselves. The problem is that while the conclusion of the original argument asserts that any physical agent limited by the PCTT can't calculate the halting problem, the argument itself presupposes a type of knowledge ─ the proof of the halting problem ─ that is not physically computable.

If Argument B implies that Argument A is false, then does this imply a solution to the question whether an alleged fact of the physical universe ─ the halting status of computation $D('D')$ ─ can be meaningfully represented by some physical computation? Argument B in itself doesn't offer a solution. It only shows that there is something wrong with Argument A. But then how could these contradictions be resolved?

Since the PCTT doesn't logically imply physicalism, it's possible that the proof of the halting problem can be a kind of platonic truth, while at the same time it's possible to hold the PCTT as true. However, this interpretation ─ while popular ─ doesn't go well with the original intent of Lloyd's argument for physical predictors. It is also not possible to imagine a hierarchy of universal Turing machines, where M_{n+1} would be able to calculate that M_n can't calculate the n-th halting problem. If this were possible, any undecidable problem could be solved as a consequence. There is also the question of solving the problem by guesswork. This in itself would not be impossible, but after a certain finite number of steps, a computer that makes a specific decision 0 or 1 ─ and thus produces a guess in any case ─ cannot guess the outcome of *any* other computer's decision, if it must make it after any *unspecified* number of steps. All these attempts show that it is not so easy to resolve the contradiction between Argument A and a physicalist-empiricist world view.

By analyzing the implications of the Physical Church–Turing thesis and its connection to the halting problem, several questions emerge. Can we assess the validity of Argument B? Were there similar agnostic arguments to Argument A? If yes, then what was the reaction to these arguments? What conclusions were reached by those similar arguments? Was there any resolution of agnosticism?

László Kalmár's critique of Church's Thesis (Kalmár 1959) presents a parallel discussion about a similar recognition of agnosticism, questioning the boundaries of effective calculability and challenging the thesis's completeness.

4. Kalmár's argument against Church's thesis

In his essay *'An Argument Against the Plausibility of Church's Thesis'* (Kalmár 1959), László Kalmár argued for the implausibility *of Church's Thesis based on a specific non-recursive function* $\psi(x)$ *of Kleene, which is defined based on a recursive function* $\phi(x,y) = 0$*. Kalmár argued that the proven non-recursive nature of Kleene's* ψ function and the Law of Excluded Middle lead to a contradiction with Church's Thesis ─ which is nothing but another name of the Church ─ Turing Thesis defined and mentioned in the former chapters. Kalmár argued that Church's Thesis is not a theorem or a definition in a mathematical sense which can be proven or disproven formally:

„In the present contribution I shall not disprove Church's Thesis. Church's Thesis is not a mathematical theorem which can be proved or disproved in the exact mathematical sense, for it states the identity of two notions only one of which is mathematically defined while the other is used by mathematicians without exact definition." (Kalmár 1959, 72)

It has to be noted first that the expression *'Church's Thesis'* denotes the same concept as the Church─Turing thesis. Alonzo Church introduced the concept of λ-calculus as a formal method to define computability (Church 1932, 1936). λ-calculus is a system for defining and manipulating functions through function abstraction and application. Independently, Kurt Gödel and later Stephen Kleene developed the concept of general recursive functions, which are constructed from basic functions using composition, recursion, and minimization (Gödel 1934, Kleene 1936). Church proposed that the class of λ-definable functions captures precisely those functions that are effectively calculable. Alan Turing, in his 1936 work, introduced the concept of Turing machines and argued that any function that can be effectively calculated by a human following a mechanical procedure can be computed by a Turing machine (Turing-computable functions). The equivalence between Turing-computable functions and general recursive functions was established through subsequent work by Gödel, Kleene, and others. The Church─Turing thesis, which emerged from these developments, asserts that the three notions ─ effectively calculable functions, λ-definable functions, and Turing-computable functions ─ are equivalent and capture the same set of functions that can be computed by any mechanical process (Copeland, 2020).

Kalmár argued that Church's Thesis should neither be considered a theorem nor a definition. He suggested that viewing it as a definition could lead to a scenario where someone might define a function that isn't effectively calculable in the terms Church proposed, yet could still be effectively calculated for specific arguments. Unlike definitions, which are irrefutable, Church's Thesis remains open to potential refutation. It can only be a 'working hypothesis'.

„For this reason, it seems [to] me better to regard such statements as Church's Thesis [...] as propositions rather than definitions, however, not mathematical but 'pre-mathematical' ones." (Kalmár 1959, 73)

Kalmár described *pre-mathematics* which *„discusses such questions as plausibility of hypotheses, adequacy of definitions etc. to be used in mathematical reasoning."* (Kalmár 1957b, translation be me)

Church's Thesis is described by Kalmár as a pre-mathematical proposition (Szabó 2018, 149) because its validity hinges on plausibility rather than mathematical proof or disproof. In highlighting its status as pre-mathematical, Kalmár underscored that discussions around Church's Thesis should focus on its likelihood of being true, rather than relying solely on mathematical arguments (Kalmár 1959, 72). Kalmár's skepticism towards the thesis stemmed from his conviction that the scope of

effectively calculable functions extends beyond the realm of recursive functions, suggesting that not all effectively calculable functions are recursive.

4.1. Kalmár's argument

Kalmár's argument is based on the non-recursive function $\psi(x)$:

$$\psi(x) = (\mu y)[\phi(x,y) = 0] = \begin{cases} \min\{y \in \mathbb{N} | \phi(x,y) = 0\} & \text{if } \{y \in \mathbb{N} | \phi(x,y) = 0\} \neq \emptyset, \\ 0 & \text{if } \{y \in \mathbb{N} | \phi(x,y) = 0\} = \emptyset. \end{cases}$$

In this formal definition, $\min\{y \in \mathbb{N} | \phi(x,y) = 0\}$ *denotes the least natural number* y *such that* $\phi(x,y) = 0$, assuming that such a y exists. If no such y exists, then $\psi(x)$ is defined to be 0. The set $\mathbb{N}$ represents the set of natural numbers.

$\phi(x,y)$ is assumed to be a general recursive or computable function. Given the focus of Kalmár's discussion is on the general recursivity and effective calculability, the specific operational details of ϕ are not elaborated. $\phi(x,y) = 0$ is the condition we're interested in. We're looking for a pair of x and y such that when they are input into ϕ, the output is 0.

The μ **operator,** also known as the minimization or least number operator in recursion theory (Davis 2003, 55-58), implies the action of finding this smallest y for which a given condition holds true. The μ–Recursion Theorem (or Minimization Principle) of computability theory (Davis 2003, 58) allows for the definition of partial recursive functions from total recursive functions. It essentially states that if we have a total recursive function $\phi(x,y)$, we can use the μ operator to define a new function $\psi(x)$ that searches for the smallest y such that $\phi(x,y) = 0$. The key aspect of unbounded minimization is that there's no predefined upper limit for y. The search continues indefinitely until a y that satisfies $\phi(x,y) = 0$ is found, or forever if no such y exists. If the minimization is unbounded, this operation may lead to $\psi(x)$ being partial, as there might not exist such a y for every x, so the application of the unbounded μ operator introduces the possibility of non-termination or undefined behavior for some inputs. $\psi(x)$ was proven by Kleene (Kleene 1936) to be non-recursive.

A key element of Kalmár's argument is that if ψ were effectively calculable, then we would have **a process** consisting of two methods to calculate it (Szabó 2018, 147)(Molnár 2016, 4-5). To calculate $\psi(p)$, we can calculate $\phi(p, 0)$, $\phi(p, 1)$, $\phi(p, 2)$... in finitely many steps until we reach the first number m, for which $\phi(p, m) = 0$, then the value of $\psi(p)$ will be m. Alternatively, to calculate $\psi(p)$, we can have a proof $\Gamma_{\psi(p)}$ which proves in a finitely many steps that there is no such m, for which $\phi(p, m) = 0$, and then the value of $\psi(p)$ will be 0. So $\psi(p)$ is effectively calculable if p is a number for which:

1. there exists such a y that $\phi(p,y) = 0$, or
2. there exists a proof $\Gamma_{\psi(p)}$ which proves that for all y, $\phi(p,y) \neq 0$.

Since Kleene proved that ψ is non-recursive, it follows from Church's Thesis that ψ is not effectively calculable. But this means none of the two methods of calculation or computation work. By following

the Law of Excluded Middle (Molnár 2016, 5), this means that there exists a number p for which $\psi(p)$ is not effectively calculable. But then this means that with this argument we proved in a finite number of steps that there exists a number p such that

1*. for all y, $\phi(p,y) \neq 0$, and
2*. there exists no proof $\Gamma_{\psi(p)}$ which proves that for all y, $\phi(p,y) \neq 0$.

So we have proved or calculated in a finite number of steps (1*) that for all y, $\phi(p,y) \neq 0$, and we have also proved or calculated in a finite number of steps (2*) that there exists no proof $\Gamma_{\psi(p)}$ which proves that for all y, $\phi(p,y) \neq 0$, which is a contradiction. Which means that either Church's Thesis or the Law of Excluded Middle must be false.

The suggestion that the ψ function could challenge Church's Thesis comes from the apparent paradox that arises from the application of the μ operator to a general recursive function ϕ. Church's Thesis equates effectively calculable functions with general recursive functions. Since Church's general recursive functions were found to be equal to the concept of computation via Turing machines (Copeland 2020), effectively calculable functions can also be equated by Turing-computable functions. The ψ function is defined in terms of a general recursive function ϕ, but when we apply the minimization operator μ, we are directed to search for the smallest y that makes $\phi(x,y) = 0$. If no such y exists, the search theoretically would go on indefinitely, making the ψ function non-computable because it would never halt. The paradoxical aspect arises when we consider that each individual computation of $\phi(x,y)$ is effective (since ϕ is a general recursive function), but the overall process of finding the minimal y might not be if there is no such y. This challenges the thesis by suggesting that even though we start with a computable function ϕ, the process of applying the μ operator leads us to a situation where we can define a function ψ that does not seem to be effectively calculable by general recursive means ─ or by a Turing machine ─, even though it is defined in terms of operations on computable functions. Eventually what Kalmár tried to show with his argument is that effective calculability can't be equated by either calculability via Church's general recursive functions or computability by a Turing machine. Kalmár argued that effective calculability was a much broader concept.

Kalmár also tried to give a more formal proof of his argument, which can be considered a variant (Kalmár 1957a, 35-37)(Kalmár 1959, 77-79). To give an outline of the proof, Kalmár began with a system of equations S that provides a general recursive definition for a function ϕ. The system S includes function symbols $\phi 1$,$\phi 2$,…,ϕ_r (with ϕ_r=ϕ) which are used in these equations. The formal system P is then defined using these function symbols, a constant symbol 0, numerical variables, the equality symbol, predicate variables, truth functions, and quantifiers. The non-logical axioms of P consist of the equations from S and the principles of the predicate calculus with identity. The inference rules of P are also those of the predicate calculus. Kalmár then suggested replacing the vague notion of proofs by arbitrary correct means with proofs within the consistent extensions of P. A consistent extension of P would include some additional formulae of P as new axioms but maintain logical consistency. If, in a consistent extension of P, we can prove a statement that asserts the non-existence of a natural number y for which $\phi(p,y) = 0$, then this statement is indeed true, and consequently, $\psi(p) = 0$. That is because if it were possible to prove $\phi(p,q) = 0$ for some natural number q, then within P and its consistent extensions, we could also prove the existence of such a y, contradicting the system's consistency if it has already been shown that no such y exists. But that is

not case, since $\phi(p,q) = 0$ can't be proven yet. Kalmár then pointed out that according to Church's Thesis, there should exist a number p for which $\phi(p,y) = 0$ for all y, and yet, this cannot be proven within any consistent extension of P. This situation would seem to contradict Church's Thesis. Kalmár's trick was then to add a new axiom to P. By adding an axiom that explicitly states $\phi(p,a) = 0$ (with a as a free variable) to P, we create a consistent extension P' where it can be proven that no y satisfies $\phi(p,y) = 0$. So in P' $\nexists(y)(\phi(p,y) = 0)$ is provable, which contradicts the implication of Church's Thesis, according to which $\exists(y)(\phi(p,y) = 0)$ is absolutely undecidable, meaning that neither $\exists(y)(\phi(p,y) = 0)$ nor $\nexists(y)(\phi(p,y) = 0)$ is decidable in any such consistent extension of P' by any Turing machine.

4.2. The Kalmár procedure

In his 1956 talk, Kalmár challenged Church's thesis with a suggested effective method we can call the Kalmár Procedure (Kalmár 1957a, 33) (Molnár 2016, 5-6). This procedure $K_{\psi(p)}$ consist of the above presented *'mechanical'* process of proof: To calculate $\psi(p)$, we can calculate with a process A $\phi(p, 0)$, $\phi(p, 1)$, $\phi(p, 2)$... in finitely many steps until we reach the first number m, for which $\phi(p, m) = 0$, then the value of $\psi(p)$ will be m. In the meantime, to calculate $\psi(p)$, we can have a proof $\Gamma_{\psi(p)}$ as a result of process B which proves in finitely many steps that there is no such m, for which $\phi(p, m) = 0$, and then the value of $\psi(p)$ will be 0. Now what Kalmár says is the following: it can't be proven that the procedure $K_{\psi(p)}$ doesn't terminate. Because if there were such a proof $\Gamma_{K\psi(p)}$, then this would mean that there exists such a p for which $\psi(p)$ is not calculable either via process A or B. But then, according process A, for all y, $\phi(p,y) \neq 0$. But this is exactly the result that we anticipated from process B. But this means that a proof $\Gamma_{K\psi(p)}$, according to which the procedure $K_{\psi(p)}$ doesn't terminate is false. There is of course no proof to accept the Kalmár Procedure as an effective procedure, but just as well, there is no proof to accept Church's Thesis either. All that Kalmár points out is that his process could just as well be a good description of an effective procedure as for example a process of computation via a Turing machine is.

4.3. Kalmár's philosophical views: Church's thesis leads to agnosticism

Kalmár perceived Church's Thesis as going beyond merely equating the intuitive concept of effective calculability with a specific, well–defined group of functions. He saw it also as endorsing a narrow interpretation that limits the broader concept of finite decision processes. Kalmár had two main contentions against Church's Thesis.

The first was that having a uniform procedure contradicts the evolving nature of mathematics and that Church's Thesis itself is fallible. If only one uniform computational process exists which is defined by Church's Thesis, then the thesis could not be fallible. Kalmár's critique of the notion of uniformity in the context of effective calculability and Church's Thesis revolves around the idea that uniformity — understood as a uniform method applicable to solve all mathematical problems of a certain type — cannot be objectively defined outside the context of our current mathematical understanding and development. This argument is encapsulated in the metaphor of a school-boy, which serves to illustrate how perceptions of uniformity can evolve with increased knowledge and understanding.

Kalmár used the metaphor of a school-boy facing diverse arithmetical problems (Szabó 2018, 150). Initially, the methods to solve these problems may not seem uniform to the student until he learns a more generalized approach, like solving equations, which unifies the methods under a broader conceptual framework. Before the advent of group theory, mathematicians worked with methods in algebra, geometry, and number theory that seemed distinct but were later understood to be unified under the group-theoretic framework. This metaphor suggests that what is considered *'uniform'* is subject to change as mathematical knowledge progresses. A procedure or method that seems to require distinct approaches at one point in time may later be seen as part of a unified method as the discipline evolves. Therefore, the notion of uniformity is relative and dependent on our current level of mathematical development. Kalmár argued that this relativity contrasts with the usually assumed objectivity of uniformity (Szabó 2018, 150-152). In traditional discussions of effective calculability and Church's Thesis, a computational method is considered effective if it is uniformly applicable across all instances of a particular type of problem. However, Kalmár pointed out that this requirement for uniformity imposes a static framework on something that is inherently dynamic and subject to change as mathematics evolves. For Kalmár, effective calculability and Church's Thesis ─ which equates the intuitive concept of effective calculability with the formal notion of recursiveness (or Turing-computability) ─ should be understood in a way that transcends the current limitations imposed by the notion of uniformity (Szabó 2018, 152). He suggested that effective calculability has an objective meaning only if it does not rely on a fixed notion of uniformity, which is bound to evolve as mathematical understanding deepens.

Kalmár's proof by arbitrary but correct means (Kalmár 1959, 74) is a vague notion that refers to the concept of establishing mathematical truths using any valid reasoning or methodology, not limited to a predefined or fixed set of rules or procedures. Kalmár argued that effective calculability and the notion of finite decision procedures should be open to any method that proves to be valid and effective, not just those that fit within the recursive framework prescribed by Church's Thesis, especially because he viewed the thesis as fallible and viewed all computably unsolvable, or absolutely unsolvable problems whose unsolvability is based upon Church's Thesis as potential falsifiers of the thesis.

„I consider any proposition stating the absolute unsolvability of some problem (with parameter) proved on the basis of Church's Thesis as a potential falsifier of all theories which are based on this thesis, of course without being sure or even without suggesting that any of them will be falsified, by solving the problem in question using some acceptable (thought of course, not recursive) method, some time in the future." (Kalmár 1967, 207)

Kalmár's second problem with Church's Thesis was based on his empiricist and physicalist views of nature stemming from his understanding of dialectical materialism (Szabó 2018, 153). Kalmár held an objectivist view according to which the statements related to arithmetic have arisen from abstractions of the properties of objects existing in the world around us (Kalmár 1957a, 34). He thought that understanding and knowledge about the objective reality ─ which he defined strictly as the *'physical world'* ─ can be progressively achieved without limit. This view has significant consequences for the field of mathematics. Kalmár argued that mathematical concepts originate from and are justified by empirical observations and experiences, linking mathematics inherently to the physical, observable world. Therefore, there's an underlying conviction within his philosophical stance that as mathematics and the sciences evolve, there will eventually be solutions to every problem, underscoring a deep connection between mathematics and the objective reality.

In a 1956 presentation at the Hungarian Academy of Sciences, Kalmár discussed Church's Thesis alongside Unsolvable Mathematical Problems, highlighting a particular way to interpret Gödel's Incompleteness Theorems and Church's Theorem (Kalmár 1957a, 33). He characterized such an interpretation as bearing 'signs of agnosticism', particularly when they were seen as evidencing the limits of human knowledge. Kalmár criticized such views, often put forth by idealist philosophers, as

agnostic, arguing that they suggest the existence of aspects of 'objective reality' that are beyond human comprehension or resolution. This starkly contrasted with Kalmár's interpretation of dialectical materialism, which posited an inherent conflict with any notion that portrays *'objective reality'* as containing unknowable or unsolvable elements. Actually, both Kalmár's 1956 talk (Kalmár 1957a) as well his 1959 paper (Kalmár 1959) can be seen as detailed arguments against such agnostic views of reality. He saw a huge concern with such views which he thought were the result of assuming Church's Thesis as true.

Kalmár specifically described the implications arising from Church's Thesis as 'strongly agnostic' (Kalmár 1957a, 29, 33). He interpreted the thesis to imply that certain properties of objective reality cannot be established through any reliable methods. This leads to a conclusion that Church's Thesis is fundamentally incompatible with the belief that humanity can eventually understand and uncover all aspects of objective reality.

„According to this, the Church's Thesis implies the assertion that there exists a law that is present in objective reality, but its existence cannot be discerned through any proper reasoning. No one who is convinced that the laws existing in objective reality can be known can accept this assertion; therefore, they cannot accept the Church's Thesis either." (Kalmár 1957a, 34, translation by me)

Connecting his general view to his particular argument against Church's Thesis, Kalmár pointed out that within his critique of Church's Thesis, it is pivotal that the function ϕ can be identified as an elementary function. This was crucial for him to assert that his arguments hold a significant connection to 'objective reality.' He explained that *„every proposition (such as $\phi(p,y) = 0$) that contains only nonnegative integers and the basic arithmetic operations (addition, subtraction, multiplication and division) refers to quantitative relations in objective reality."* (Kalmár 1957a, 34)

The concrete problem with Church's Thesis arises if there exist **absolutely undecidable propositions**. According to Kalmár, $\exists (y)(\phi(p,y) = 0)$ can be considered as such, because it can not be proved, because it is false, but its negation can't be proved either, since if proved, is would yield the value of 0 for $\psi(p)$. In Kalmár's words, it is absolutely undecidable for the following reason: *„As a matter of fact, the problem, if the proposition in question holds or not, does not contain any parameter and, supposing Church's Thesis, the proposition itself can be neither proved nor disproved, not only in the frame of a fixed postulate system, but even admitting any correct means."* (Kalmár 1959, 75)

The problem with such an absolute undecidability of a proposition is twofold. On the one hand, Kalmár thought that *„this 'absolutely undecidable proposition' has a defect of beauty: we can decide it, for we know, it is false".* (Kalmár 1959, 75) So it is decidable, at least for humans. On the other hand, the absolute undecidability of this proposition cannot be proved *„by any correct means".* (Kalmár 1959, 75) (Kalmár 1957a, 31-32) That's because a proof of its undecidability would show that there is no such y for which $\phi(p,y) = 0$ holds, which at the same time would prove $\exists (y)(\phi(p,y) = 0)$ and then this would also entail that $\psi(p) = 0$.

The real problem is that we have ϕ, which is an elementary function, **which expresses a basic truth about physical reality** (Kalmár 1957a, 33-34)**.** But then the function ψ, which is based on ϕ, also expresses a truth or a property about physical reality, but it can't be proved by any correct means if Church's Thesis is assumed.

„Church's Thesis contains the claim that there is a certain T statement that is, on one hand, true, and on the other hand, cannot be proven (nor disproven) through any proper reasoning; specifically, a T

statement of the form that there does not exist any y non-negative integer for which $\phi(n,y) = 0$, where ϕ is some general recursive, or even elementary function, and n is some non-negative integer. This claim inherent in Church's Thesis is strongly agnostic, even of a Kantian nature, since it states that the T statement is true in objective reality itself (an sich), but it can never be revealed to us that it is so, since we could only know this through some form of proof." (Kalmár 1957a, 34)

Kalmár's argument in his 1959 article can be seen as an attempt to show that any correct proof of the absolute undecidability of $\exists (y)(\phi(p,y) = 0)$ leads to the above presented contradiction with either Church's Thesis or the Law of Excluded Middle.

4.4. Mendelson's critique of Kalmár's argument

Mendelson (Mendelson 1963) criticized Kalmár's argument, because it relies on the implicit assumption that the set of correct proofs is effectively enumerable ─ meaning that the correct proofs are enumerable by a Turing machine or an equivalent effective process. This implicit assumption, denoted by Mendelson as Hypothesis H (Mendelson 1963, 203), is crucial for Kalmár's argument that there must exist some natural number p for which it is not possible to prove, by arbitrary correct means, that no y satisfies $\phi(p,y) = 0$, and therefore, for this $\psi(p)$ would be not effectively computable. Mendelson argued that without this assumption of effective enumerability of the set of correct proofs, Kalmár's conclusion does not necessarily follow. This is because the ability to enumerate all proofs by arbitrary correct means would be required to ensure that for any p, either a y such that $\phi(p,y) = 0$ can be found, or it can be proved that no such y exists. If this set of proofs can not be effectively enumerated, then **the process** described by Kalmár for determining the value of $\psi(p)$ for every p cannot be guaranteed to terminate, thereby challenging Kalmár's assertion that this leads to a contradiction with Church's Thesis.

A key element of Kalmár's argument is that if ψ were effectively calculable, then we would have **a process** consisting of two methods to calculate it, but also that if ψ is **not** effectively calculable according to Kleene's proof, then Kleene's proof implies that we calculated in a finite number of steps that there exists a number p such that:

1. for all y, $\phi(p,y) \neq 0$, and
2. there exists no proof $\Gamma_{\psi(p)}$ which proves that for all y, $\phi(p,y) \neq 0$.

But if Kleene's proof is not recursively enumerable or if there exists no effectively enumerable proof or a calculation for any of these two conclusions, then we have not proved that for all y, $\phi(p,y) \neq 0$, and we have also not proved that there exists no proof $\Gamma_{\psi(p)}$ which would prove that for all y, $\phi(p,y) \neq 0$. Which means that it is not true that either Church's Thesis or the Law of Excluded Middle must be false, so that there is no contradiction if the set of correct proofs is not recursively enumerable (Mendelson 1963, 203).

Kalmár's process description yields the following result: for any p, either a y such that $\phi(p,y) = 0$ can be found, or it can be proved that no such y exists. Kalmár argued the following way. If $\exists (y)(\phi(p,y) = 0)$ can be decided such that for $\phi(p,q)$ the property holds, then this q can be found. Kalmár thought that $\exists (y)(\phi(p,y) = 0)$ being decidable in this case is a consequence of q being found.

Kalmár also thought that if $\exists (y)(\phi(p,y) = 0)$ can not be decided because for some q in $\phi(p,q)$ the property doesn't hold, then this q can not be found and **this must have a proof,** so therefore it would also be decidable. In short: either we have a proof of $\exists (y)(\phi(p,y) = 0)$ or we have a disproof of $\exists (y)(\phi(p,y) = 0)$ and there is no third option. This is a strong assumption about the decidability or computability of all mathematical propositions, which implicitly relies on an effectively enumerable set of proofs or computational methods. Mendelson's critique highlighted the issue with this assumption. Without the effective enumerability of the set of correct proofs, there could indeed be a third option where the process does not terminate because neither a satisfying y is found nor is it possible to prove that no such y exists. This situation would leave some propositions about the existence of y undecidable, challenging Kalmár's assertion of universal decidability based on the process he describes. This implies that Kalmár's argument critically hinges on an optimistic view of decidability and computability, as Mendelson pointed out (Mendelson 1963, 204).

The question of whether the set of correct proofs is recursively enumerable has been a central issue in mathematical logic and the philosophy of mathematics. In formal systems like Peano Arithmetic (PA), the set of correct proofs ─ those that adhere to the syntactical rules of the system and conclude with valid theorems ─ is considered recursively enumerable. That's because proofs in formal systems are finite sequences of symbols following specific syntactic rules. This means that, in principle, a Turing machine could systematically generate all such proofs, given enough time. However, Mendelson pointed out that this assumption might be too optimistic because it presumes that every correct proof can be found by some effective process. In reality, our ability to effectively enumerate all correct proofs may be limited by various factors, such as the complexity of the proofs or the nature of the mathematical systems in question.

4.5. Analysis of Kalmár's argument with the halting problem

We can analyze Kalmár's and Mendelson's argument with more insight if instead of Kleene's ψ function, we adapt the halting function to a similar formal description to what is used for ψ. Suppose the following function $G(x,y)$, where the variable x represents both the x-th Turing machine in a standard, Gödel-number based enumeration of Turing machines and also the x-th input to that Turing machine, and the variable y represents a fixed number of computational steps for the x-th Turing machine on the x-th input. Then $G(x,y)$ can be defined as follows:

$$G(x,y) = \begin{cases} 1 & \text{if the } x\text{-th Turing machine on input } x \text{ halts within } y \text{ steps,} \\ 0 & \text{if the } x\text{-th Turing machine on input } x \text{ does not halt within } y \text{ steps.} \end{cases}$$

$G(x,y)$ is recursive because the number of computational steps is finite. It can be considered as the finite version of the halting function. Based on the total computable $G(x,y)$, we can define the non-computable general halting function as follows:

$$H(x) = \begin{cases} \mu(y)[G(x,y) = 1] & \text{if } \exists y[G(x,y) = 1], \\ 0 & \text{if } \neg\exists y[G(x,y) = 1]. \end{cases}$$

The first case of $H(x)$, $\mu(y)[G(x,y)=1]$, uses the minimization operator μ to find the smallest y – the least number of steps – for which $G(x,y)$ is 1. If such a y is found, it indicates that the Turing machine x does halt on input x, and $H(x)$ will be equal to that smallest y. The second case, where $H(x)=0$, addresses the situation where no such y exists. This means that for all natural numbers y, $G(x,y)$ is never 1, so the machine x never halts on input x. Here, $H(x)$ is 0, indicating non-halting. So $H(x)$ discerns whether a Turing machine halts on a given input and, if it does, provides the least number of steps required to reach the halting state. If the machine does not halt, $H(x)$ simply returns 0. We can then highlight the result of Turing's proof in this formal notation by using $'D'$ as the specific Gödel-number of the anti-diagonal function D:

$$H('D') = \begin{cases} \mu(y)[G('D', y) = 1] & \text{if } \exists y[G('D', y) = 1], \\ 0 & \text{if } \neg\exists y[G('D', y) = 1]. \end{cases}$$

What can be learned from Kalmár's argument and Mendelson's critique combined, is that **if the set of correct proofs were recursively enumerable**, then if H were effectively calculable, then according to Kalmár, we would have **a process** consisting of two methods to calculate it. To calculate $H(p)$, we could calculate $G(p, 0), G(p, 1), G(p, 2)...$ in finitely many steps until we reach the first number m, for which $G(p, m) = 1$, then the value of $H(p)$ will be m. Alternatively, to calculate $H(p)$, we can have a proof $\Gamma_{H(p)}$ which calculates or proves in a finitely many steps that there is no such m, for which $G(p, m) = 1$, and then the value of $H(p)$ would be 1. So $H(p)$ would be effectively calculable if p is a number for which:

3. there exists such a y that $G(p,y) = 1$, or
4. there exists a proof $\Gamma_{H(p)}$ which proves that for all y, $G(p,y) \neq 1$.

Since Turing proved that H is non-recursive, it follows from Church's Thesis that H is not effectively calculable. But this means none of the two methods work. By following the Law of Excluded Middle, this means that there exists a number p for which $H(p)$ is not effectively calculable – which in case of Turing's proof would be the Gödel-number $'D'$. But then this means that if Turing's proof is considered as 'correct' and we also hold that the set of correct proofs is recursively enumerable, then this proof can be included in the enumeration of correct proofs and we have proved or calculated in a finite number of steps that there exists a Gödel-number $'D'$ for which:

1. for all y, $G('D',y) \neq 1$, and
2. there exists no proof $\Gamma_{H('D')}$ which proves that for all y, $G('D',y) \neq 1$.

So we have proved in a finite numbers of steps that for all y, $G('D',y) \neq 1$, and we have also proved in a finite numbers of steps that there exists no proof $\Gamma_{H('D')}$ which proves that for all y, $G('D',y) \neq 1$, which is a contradiction. Which means that given the assumption that the set of correct proofs are recursively enumerable, either Church's Thesis or the Law of Excluded Middle must be false.

4.6. Summary of Argument K

Kalmár's argument combined with Hypothesis H along Mendelson's critique (Kalmár 1959, 74) (Mendelson 1963, 203-204) ─ we can call it Argument K ─ can be summarized as follows: We assume Church's Thesis (or in other words, the Church─Turing thesis), according to which the effectively calculable functions are the recursive functions. We assume that the set of correct proofs are recursively enumerable. We also assume ─ following Kleene's proof or following Turing's proof ─ that there exists a function ψ ─ or a function H ─ which is non-recursive, which is therefore not effectively calculable or computable. Then this leads to a contradiction, because this implies that on the one hand we can provably compute that a statement such as $\exists (y)(\phi(p,y) = 0)$ ─ or $\exists (y)(G(p,y)) = 1)$ ─ is not effectively computable and that on the other hand there is no computable proof that such statements are not effectively calculable.

Kalmár argued that because ϕ can be an elementary function, then $\exists (y)(\phi(p,y) = 0)$ expresses a basic truth about physical reality. If this were the case and according to Kleene's proof $\exists (y)(\phi(p,y) = 0)$ would not be Turing-computable and according to Church's thesis it is exactly the Turing-computable functions which are effective, then it would lead to world view where a basic truth about physical reality could not be effectively obtained. According to Kalmár's views, this would contradict physicalism and empiricism, because Kalmár thought that in physical reality all truths should be effectively obtained.

4.7. Argument K and Argument B

Argument K is actually very similar to Argument B, which was the following: We assume the Church─Turing thesis. We assume physicalism. We also assume ─ following Turing's proof ─ that there exists a program H, which is not computable. This leads to a contradiction, because on the one hand, this implies that a physical and therefore computational agent M could calculate that a computation $H('D','D')$ is not computable. On the other hand, by following Turing's proof, if such computation by M would be obtainable by H as a subroutine, that would eventually contradict that $H('D','D')$ is not computable.

There are two differences between Argument K and Argument B. One is the application of the assumption in Argument K that the set of proofs are effectively enumerable versus the assumption of physicalism in Argument B. Actually, from the point of the arguments, they are not so different. If we hold on to the assumption of physicalism in Argument B, then all proofs must be finite, physical processes. Proofs by definition are finite processes, which lead from a finite set of axioms and derivation rules through a sequence of formula-transformations to the final formula which is the result of the proof (Crossley et. al. 1990, 14). In a physicalist view of the world, any real proof can and must be a physical process, therefore any real physical proof must be effectively enumerable by a physical computation. But this implies that if Turing's proof is considered as 'correct' and we also hold that the set of correct proofs is recursively enumerable, then this proof would be included in the enumeration of correct proofs which then eventually leads to a contradiction. This shows that it is the enumerability of proofs by computations which causes the contradiction between the PCTT and the halting problem, but this enumerability is a necessary consequence of physicalism and the PCTT. It also follows that Kalmár's argument combined with Mendelson's Hypothesis H underscores and strengthens Argument B.

It must be remarked that physicalism is also in line with Kalmár's thought that all elementary function express some basic property of physical reality. Although Mendelson was right in saying that the Kalmár's argument does not hold against Church's Thesis (because it's possible that all correct proofs are not effectively enumerable), Kalmár's argument actually does stand when combined with Church's Thesis and physicalism combined, where every *physically* correct proof would be included in the enumeration of correct proofs.

The second difference between Argument K and Argument B is that Argument K is used by Kalmár explicitly to argue against Church's Thesis as the definitive source of the contradiction. But it is quite far from being evident that it is a variant of Church's Thesis, which is the source. Actually, there are three assumptions in each argument, which are contradictory and therefore each of them can be responsible for the contradiction: 1. a variant of the Church–Turing thesis; 2. a variant of an uncomputable function; 3. a variant of the assumption that the set of correct proofs are effectively enumerable. If the question is if it is possible to hold a physicalist worldview without contradictions and we assume that in a physicalist setting, all physically correct proofs are enumerable by physical-computational agent, then this implies that there must be something wrong with either the variant of the PCTT or the variant of an uncomputable function, or both.

Argument B implies that even Turing's proof that the output value of $H(\text{'D'}, \text{'D'})$ can't be computed is also a statement which can't be computed by any physical computation. This implies that no physical validator could determine whether $H(\text{'D'}, \text{'D'})$ is computable or uncomputable. But if that's true and at the same time there is no solution on how to compute or determine $H(\text{'D'}, \text{'D'})$, any such solution would just deepen the agnosticism, as the question still remains whether $H(\text{'D'}, \text{'D'})$ is computable or not. However, Argument B in itself doesn't offer a solution. It only shows that there is something wrong with Argument A. This suggests that it might be the naive application of the PCTT, which disregards what is physically meaningful, which disregards any physical constraints on measurement and which applies an unbounded prediction which can be the source of contradiction of Argument A with physicalism and empiricism. Based on this conjecture, we have to analyze the meaningfulness of physical computations, particularly focusing on the empirical constraints that such physical meaningfulness requires.

5. Computations carrying meaning with empirical constraints

5.1. Conditions for establishing meaning in physical formal systems

Can a physical computer executing a specific program H represent the halting problem? In order to analyze this question, we would need a physicalist definition of representation and meaning of computations. In this case, we can apply and adapt the definition of representation or meaning (E. Szabó 2017, 8-9) for physical formal systems. Because of the definition's physicalist background, it should be perfectly applicable to the case of representation of physical computations. This definition of meaning for physical formal systems comprises two conditions, A and B, formulated within the context of a formal system and its relationship to physical reality.

Condition A:

There exists a class of formulas $\{A\lambda\}\lambda$ in the formal language L and a corresponding class of states of affairs $\{a_\lambda\}_\lambda$ in the physical world such that for some specific λ_0, $A = A_{\lambda 0}$ (a specific formula in L) and $a = a_{\lambda 0}$ (a specific state of affairs in the physical world).

This condition establishes a requirement for a one-to-one correspondence or mapping between certain formulas in a formal system and specific states of affairs in the physical world. It implies that for a formula $A_{\lambda 0}$ to have meaning, it must be part of a $\{A\lambda\}\lambda$ class of formulas, each of which is systematically related to a class $\{a_\lambda\}_\lambda$ of physical states of affairs. However, this condition only establishes a preliminary mapping through the same λ indexing.

Condition B:

For all λ,

> if a specific state of affairs a_λ is the case (actually occurs) in the physical world, then the corresponding formula A_λ is a theorem in the formal system $(\Sigma_L \vdash \boldsymbol{A}_\lambda)$;
>
> if a specific state of affairs a_λ is not the case (does not occur), then the negation of the corresponding formula $\neg A_\lambda$ is a theorem in the formal system $(\Sigma_L \vdash \neg\boldsymbol{A}_\lambda)$.

This condition establishes a consistent and systematic relationship between the occurrences (or non-occurrences) of states of affairs in the physical world and the derivation of corresponding theorems (or their negations) within the formal system. For a formula to meaningfully represent a state of affairs, the formal system must be able to correctly correspond to the occurrence (or absence) of that state of affairs based on whether the formula (or its negation) can be derived as a theorem within the system.

Condition A establishes the basis for a meaningful discourse by stating a direct correspondence, which can be thought of as a **semantic mapping** between specific formulas within a formal system L and particular states of affairs in the physical world U. This implies that for a formula to have potential meaning, it must be part of a structured relation that systematically associates elements of the formal system with elements of physical reality.

Condition B takes this a step further by asserting that it's insufficient for this mapping or interpretation to merely exist; **it must also be empirically validated** for any specific corresponding elements a_λ and A_λ. In other words, the relationship established by Condition A must accurately reflect the way the world works: if a state of affairs is the case in the physical world, then the corresponding formula should be derivable within the formal system, and vice versa.

'For all λ' implies that these conditions (both the mapping and its correctness evaluation) **must apply across a whole class of states of affairs and their corresponding formulas**, not just for isolated or individual cases. This universality implies that the meaning derived from this semantic mapping is systematically applied and consistently holds true across different instances within the formal system.

The remark about G-sentences not being able to participate in the semantic construction due to the requirement for for all A_λ in $\{A\lambda\}\lambda$ that either $\Sigma_L \vdash \boldsymbol{A}_\lambda$ or $\Sigma_L \vdash \neg\boldsymbol{A}_\lambda$ is the case is **critical**. This requirement establishes that the formal system can produce a definitive truth value for each formula that represents a state of affairs in the universe U. G-sentences (E. Szabó 2023, 5), by their nature, contradict this requirement because they do not have a clear truth value within the system ─ neither they nor their negation are not provable within the system, so this condition excludes formulas which can't be proved by the formal system from being meaningful. Taking into account that according to physical realism, the states of affairs in the physical world are considered either definitely true or definitely false, this means that the condition of meaning is such that it requires both the possibility of an occurrence and non-occurrence of a correlation between the formal system L and the universe U. To clarify with an example, consider a state of affairs a_λ in U and a corresponding formula A_λ in L. The meaning conditions ensure that:

if a_λ were to occur (be the case), then A_λ must be provable ($\Sigma_L \vdash \boldsymbol{A}_\lambda$);
if a_λ were not to occur (not be the case), then $\neg A_\lambda$ must be provable ($\Sigma_L \vdash \neg\boldsymbol{A}_\lambda$).

This relationship does not necessarily mean each formula within L individually embodies counterfactual scenarios, but rather that the formal system L, through its relationship with U and the conditions of meaning, can adequately reflect the counterfactual structure of physical states of affairs..

If semantic construction were to include G─sentences without any qualifications, it would mean that there exist scenarios (reflected by G─sentences) where the formal system L cannot provide a provable formula A_λ or its negation $\neg A_\lambda$ to correspond to whether a particular state of affairs is the case or not in U. This would create a gap in the counterfactual completeness of the correlation between L and U, as the formal system would fail to account for certain states of affairs in a manner that aligns with the condition of meaning defined by the provability of A_λ or $\neg A_\lambda$.

5.2. Determinability of counterfactual completeness

A condition of meaning from the part of the formal system must be subject to either empirical verification or falsification. This is an implicit empiricist criterion, which requires *that it could have been otherwise* and determinably so. It could have been that there is no correlation between U and L and in any such case, this counterfactual possibility must be empirically verifiable. This introduces the notion of contingency or the possibility of alternate outcomes, because this implicit condition

emphasizes that the correlation between L and U presupposes that L must be able not just to derive either A_λ or $\neg A_\lambda$ as a theorem, but that this must be determinably so, because this capacity must be either empirically verifiable or falsifiable.

To provide a basic example to explain the difference between a simple requirement of counterfactual completeness versus the deeper, empirical requirement of determinability of counterfactual completeness, we can consider the following example.

Suppose the following **Case 1, which features counterfactual completeness without empirical determinability**. Imagine we have a theoretical system L that, based on a complex set of physical laws, initial conditions, and theoretical constructs, predicts the outcome of every coin flip. This system **claims** to account for all possible outcomes: heads (A) or tails ($\neg\mathrm{A}$) for any given flip, making it counterfactually complete. However, suppose the system's complexity and reliance on unobservable or undefined theoretical constructs (e.g., 'quantum coin states' that don't correspond to any measurable quantum property) mean that its predictions cannot be empirically determined. We cannot measure or observe these *'quantum coin states'* directly to verify the system's predictions. Even though L is supposed to be capable of predicting the outcome of any coin flip, making it counterfactually complete, we have **no empirical method** to confirm or refute its predictions. The system's claims remain speculative and untestable, disconnected from practical reality. In contradiction to Case 1, **we can imagine Case 2**, which features **empirical determinability of counterfactual completeness.** Consider another system L' that predicts coin flip outcomes based on physically observable factors like the coin's initial position, flipping force, air resistance, and so on. This system also aims to be counterfactually complete, offering predictions for heads (A) or tails ($\neg A$) under any given class of initial conditions. In this scenario, the factors the system relies on are measurable and observable. We can set up experiments to precisely control and measure these factors, then flip the coin under those conditions to see if the outcome matches the system's prediction. For any given prediction made by L', we can empirically test whether the prediction accurately corresponds to the actual outcome of the coin flip. In this case, the system's claims are not speculative, but they describe actual states of affairs.

5.3. Meaning carrying computations

To provide a physical definition of meaning for computations, we have to first define computer programs as physical objects existing in the physical world. This is an unusual treatment of computer programs, but in physical reality, every computer is a physical object, every computer program is nothing but a specific configuration of physical objects and the computations that a computer executes are nothing but special, dynamically changing configuration of physical objects, whose seemingly abstract nature can be explained without any reference to a platonic realm, similarly to physical formal systems (E. Szabó 2017, 5). In this context, a computer program is conceptualized not merely as an abstract sequence of instructions but as a physical manifestation of those instructions within a computing device. A computer is a physical device designed to process data, consisting of hardware components like processors, memory, and storage that perform calculations and store information. It is a specific configuration of physical states. A computer program is a sequence of instructions aimed at performing a specific task, physically represented within the computer's hardware as configurations of data. Inputs are the data that a computer or program processes, physically represented within the computer and manipulated according to the program's instructions. Outputs are the results generated by a computer program after processing inputs, physically manifested in various forms such as displayed information or stored data. Computation, in this context, is the process in which a computer executes a program using specific inputs to generate

outputs. This process encompasses the physical actions of reading, processing, and transforming data through the sequential execution of the program's instructions. Computation P refers to the specific instance where a physical computer executes a given computer program P on certain inputs to produce outputs.

The conditions for meaning for formal systems outline a definition of how a formula within a formal system can represent or mean a state of affairs in the physical world, under the conditions (a) and (b). This is an attempt to integrate the concept of meaning within a physicalist framework, suggesting that meaning arises from a correspondence between the behavior of formal systems and the states of affairs in the physical world. To translate this into the realm of computer programs, we have to consider some key points. According to the physicalist perspective adopted, formal systems are not abstract entities but are physical objects. This means that the formal system itself, including the derivations it performs, is a part of the physical world. A formula within a formal system is said to mean or represent a state of affairs if there's a correlation between the formula being derivable within the system and the occurrence of the corresponding state of affairs in the physical world. Translating this to computer programs involves recognizing that in the context of computer programs, computations performed by a program can be seen as analogous to the derivations in a formal system. A computation or a series of computations would carry meaning if they correspond to or predict states of affairs in the physical world. Computer programs can be viewed as embodiments of formal systems, where the code defines a set of rules and operations that can be executed mechanically, akin to the derivation rules in a formal system.

In this adaptation, both computations and programs carry meaning, but in slightly different contexts. A specific computation or the outcome of a program (for example the result of a simulation) represents a state of affairs. The computation carries meaning in that it can be correlated with physical events or conditions, satisfying condition (b). The program itself can be seen as the broader formal system. It's the structure within which computations occur and meanings are assigned. The design of the program ─ how it structures data, performs operations, and what outputs it generates ─ embodies the rules and axioms of a formal system. Therefore, in this adaptation to the realm of computer programs, both the specific computations (as instances of deriving meaning) and the programs (as the overarching formal systems) carry meaning. The crucial aspect is the real correlation between the computations a program performs and the states of affairs in the physical world. Thus, it will be computation P, the entire process of a physical computer executing a specific program P on a specific input that carries meaning. Based on this, we can construct a definition of meaningful computation as follows.

Condition A for Meaning Carrying Physical Computations

For a computation P to carry meaning, there must exist a systematic mapping between P's inputs and outputs and specific states of affairs in the physical world U. Specifically,

> There exists a class of inputs $\{i_\lambda\}_\lambda$ that the computation P can process, and a corresponding class of outputs $\{o_\lambda\}_\lambda$ that the computation P produces, such that each input-output pair $\{i_\lambda$, $o_\lambda\}$ is systematically associated with a state of affairs a_λ in the physical world U.

This condition requires that the computation's functionality is not arbitrary but systematically related to the physical world by a systematic mapping. The computation must be designed in such a way that its input and output have clear correspondences with real-world phenomena or states of affairs. In the context of meaningful computation as defined, where the computer's inputs and outputs are mapped to states of affairs in the physical world, the term *'state of affairs a_λ'* refers to both initial conditions as well as the result of those conditions. So inputs to the computation represent data or initial conditions in the physical world, while outputs represent the computation's conclusions or predictions of those

inputs. The meaning of a computation is rooted in how these inputs and outputs relate to actual states of affairs, how the computation interprets the world and acts upon it. Interpreted this way, **a computer executing a program mirrors or models deterministic causal processes** of the physical world by systematically transforming inputs, representing initial conditions, into outputs, or effects, through computations that simulate the cause-and-effect relationships observed in nature. Each input-output pair is linked to a specific state of affairs in the physical world, and the computer's operations have relevance to real-world conditions and scenarios.

Condition B for Meaning Carrying Physical Computations

A computation P carries meaning, if for all λ,

> if $\{a_\lambda\}$ denotes a state of affairs in U which is the case, then the corresponding computation P on input $\{i_\lambda\}$ halts on output $\{o_\lambda\}$ with a correct value;
>
> if $\{a_\lambda\}$ denotes a state of affairs in U which is not the case, then the corresponding computation P on input $\{i_\lambda\}$ halts on output $\{o_\lambda\}$ with a correct value.

Specifically for decision problems which require a yes / no answer to an infinite class of questions, we can align the definition the following way. A computation is carries meaning, if for all λ,

> if $\{a_\lambda\}$ denotes a state of affairs in U which is the case, then the corresponding computation P on input $\{i_\lambda\}$ halts on output $\{o_\lambda\}$ with a value of $1;$
>
> if $\{a_\lambda\}$ denotes a state of affairs in U which is not the case, then the corresponding computation P on input $\{i_\lambda\}$ halts on output $\{o_\lambda\}$ with a value 0.

Defined this way, a non-meaningful general computation P is one which either doesn't halt, or doesn't halt with a correct output value $\{o_\lambda\}$ on **at least one** such input $\{i_\lambda\}$ corresponding to a specific state of affairs $\{a_\lambda\}$ in U. A non-meaningful singular computation P on a specific input $\{i_\lambda\}$ is one which either doesn't halt, or doesn't halt correctly on the corresponding specific state of affairs $\{a_\lambda\}$ in U.

Alternatively, for a given decision problem represented as a class of questions Q, a computation carries meaning if it systematically processes each question q in Q according to the following criteria:

> if a question q from Q corresponds to a state of affairs in the universe U that is the case (the correct answer is $'YES'$), then the corresponding computation P on input i_q halts with an output o_q signifying a value of $1;$
>
> if a question q from Q corresponds to a state of affairs in U that is not the case (the correct answer is $'NO'$), then the corresponding computation P on input i_q halts with an output o_q signifying a value of 0.

This definition aligns with the notion of decidability for a class of questions, emphasizing that meaningful computation is achieved when there exists a clear, algorithmic method (embodied by computation P) that can, for each question q within the class Q, accurately determine the truth or falsity of the corresponding state of affairs in U. Such a computation P must operate uniformly across

the entire class Q, thereby it establishes the class's decidability by ensuring a consistent and finite process for reaching a decision on every question q it encompasses.

5.4. Meaning carrying simulating computations

By adapting the concepts to the special case where one computation $C2$ meaningfully represents or simulates another computation $C1$, we can create a definition incorporating both conditions A and B as follows.

Condition A for Meaning Carrying Computation Representations

> (a) There exists a class of input configurations $\{i\}$ for computation $C1$ and a corresponding class of input configurations $\{j\}$ for computation $C2$, such that for some specific λ_0, an input configuration $i = i_{\lambda 0}$ (a specific input for $C1$) corresponds to an input configuration $j = j_{\lambda 0}$ (the associated specific input for $C2$);
>
> (b) There exists a class of expected output configurations $\{o1\}$ from $C1$ and a corresponding class of expected output configurations $\{o2\}$ from $C2$, such that for the specific λ_0, an output configuration $o1 = o1_{\lambda 0}$ (a specific expected output from $C1$) corresponds to an output configuration $o2 = o2_{\lambda 0}$ (the associated specific expected output from $C2$).

This setup implies that for every selected pair of corresponding input configurations between $C1$ and $C2$, there is an anticipated association of their outputs, reflecting a systematic approach to how $C2$ is expected to replicate computational outcomes of $C1$.

Condition B for Meaning Carrying Computation Representations

A physical computation $C2$ represents computation $C1$ if, for all λ,

> If computation $C1$, for a given input i_λ, halts and produces an output $o1_\lambda$, then the representing computation $C2$, when given the corresponding input j_λ, must also halt and produce a corresponding output $o2_\lambda$.

Specifically, a physical decision computation $C2$ represents computation $C1$ if, for all λ,

> if, for a given input $\{i_\lambda\}$, computation $C1$ halts with an output of 1 then the representing computation $C2$, when given the corresponding input $\{j_\lambda\}$, must also halt and produce an output of 1;
>
> if, for a given input $\{i_\lambda\}$, computation $C1$ halts with an output of 0, then computation $C2$, when given the corresponding input $\{j_\lambda\}$, must also halt and produce an output of 0.

The above approaches explicitly cover both scenarios, where a computation represents **physical states of affairs** and where one computation represents the **computations of another computation**. In physical world representations it ensures that the computer's computation is an

accurate reflection of real-world conditions, while in computation-to-computation representations, it ensures that one computation can serve as a faithful replica of another's computations. In short, this can be considered as a unified approach.

5.5. The requirement of halting

A non-halting program (one that runs indefinitely without reaching a conclusion) does not provide a result that can be assessed for correctness. Therefore, for a computation P on an input i to be correct, it needs to halt first, meaning it must complete its execution within a finite number of steps. In the case of meaningful computation representation, this formulation of Condition B explicitly focuses on the binary halting outcomes of computations $C1$ and $C2$, ensuring that $C2$ meaningfully represents $C1$ by mirroring its halting behavior with identical outcomes for all given inputs. non-halting cases are considered outside the scope of this meaningful representation, emphasizing the focus on halting with determinate outcomes (1 or 0).

In the original definition of a meaningful physical formal system (E. Szabó 2023, 5), the remark about G-sentences not being able to participate in the semantic construction due to the requirement for each A_λ to either be provably true ($\Sigma_L \vdash \boldsymbol{A}_\lambda$) or provably false ($\Sigma_L \vdash \neg\boldsymbol{A}_\lambda$) was expressed as critical. Translating this to the context of computation, particularly with reference to Turing's proof of the undecidability of the halting problem, we encounter a parallel scenario. Computations that are meaningful, according to this definition, must halt and return a definitive output $(0$ or $1)$ that correctly indicates whether a stated condition is true or false in the universe U. However, computations like the one described by the halting problem (e.g., determining whether $H(D, D)$ halts), inherently lack this definitive outcome by either not halting or not providing a 'correct' outcome in the sense of aligning with the truth or falsehood of a state of affairs. This means that computations $P(i)$ which either don't halt or don't halt correctly can't participate in such a semantic construction, therefore the results of such computations can't be meaningful.

5.6. Example 1

We can describe an example decision problem $UP1$ as follows:

P executes the program of Peano Arithmetics (syntax, axioms, derivations rules, etc).
U is the physical universe of apples in which apples can be counted, added, multiplied, etc.
In this U, everything holds that should hold for the Natural Numbers $\mathbb{N}$, but instead of them it holds for Apples.

$UP1$ is the Decision Problem as follows:

Let $UP1$ be the problem of determining whether it holds for all (a_λ) collections of apples that it is decidable that a given collection (a_λ) is a prime number of apples.

Formally, we can express $UP1$ as:

$UP1\ \{a_\lambda\}_\lambda$ = *'Is it true for all* $\{a_\lambda\}_\lambda$ *that* a_λ *is a prime number of apples?'*

This problem represents an infinite class of questions regarding the class of apples in the physical world. The singular instances of the problem regard individual collections of physical apples (a_λ). The corresponding inputs i_λ of the computation P (io_λ) are interpreted as calculating a result to $UP1$ are physical tape symbols. The result of the problem and the output of the corresponding computation P interpreted as calculating a result for $UP1$ is a boolean binary value: either 0 or 1, meaning false or true.

Condition A for Computation P:

Each input i_λ represents a specific natural number of apples a_λ to be evaluated for primality. For each input i_λ, computation P determines if a_λ is a prime number of apples and outputs 1 (true) if a_λ is prime, and 0 (false) if a_λ is not prime. Each input-output pair (i_λ, o_λ) of the computation P directly corresponds to whether the physical collection of apples a_λ cannot be arranged in more than one row in such a way that there are the same number of objects in every row, but more than one object.

Condition B for Computation P:

For all instances a_λ:

> If the collection of apples a_λ cannot be evenly distributed into multiple rows with more than one apple per row, then $P(i_\lambda) = 1$.
>
> If the collection of apples a_λ can be evenly distributed into smaller equal-sized groups, then $P(i_\lambda) = 0$.

In order to explain Condition A within this example, we can see that the physical computation P is interpreted as to compute if *'*a_λ *is a prime number of apples?'* which we call the problem $UP1$. In this case, a semantic mapping involves creating a mapping between the symbols on the tape of the computer P, such that for example in a unary notation a collection of 3 apples is assigned to $n+1$ number of 1 marks on the computer's tape. But this is just a preliminary mapping. This step is crucial but, by itself, doesn't assign the symbols with any 'meaning' beyond their representation as numbers in a particular notation. It's a necessary but not sufficient condition for establishing a semantic mapping. On top of this mapping, according to Condition B, a meaningful computation also involves executing a computation P such that it actually calculates whether a_λ is a prime number of apples. Simply mapping the marks 1111 to 3 apples would involve nothing like a meaningful semantic mapping in the sense of Condition B. In order for Condition B to hold, the whole computation P must be mapped so that it actually, verifiably computes the given problem.

The essence of semantic mapping, particularly in the context of computations, involves not just representing data but also encoding a procedure or algorithm that operates on this data to solve a specific problem. The computation, for example in the case of a physical Turing machine, includes a class of states and transition rules that guide the machine's head to move across the tape, read symbols, write symbols, and change states in a way that, given the representation of a number of apples a as input, halts with an output that correctly indicates whether a is prime number of apples.

For the physical computation P to genuinely compute whether a_λ is a prime number of apples, every aspect of its operation — how it represents numbers, how it manipulates those representations, and how it arrives at a conclusion — must align with the mathematical definitions and procedures for primality and also with the physical world of apples. This means that both the representation scheme (e.g., unary notation) and the computational procedure (the computer program) must be correctly designed and implemented.

The condition of the determinability of counterfactual completeness can be fulfilled here, because the states of affairs in U regarding the primality of a collection of apples are expected to be either true or false and the computation P halts on some output value. Following the philosophical reflections of Kalmár (Kalmár 1957a, 34), such primality of apples in the physical world simply means that the apples in the specific collection a_λ cannot be arranged in more than one row in such a way that there are the same number of objects in every row, but more than one object.

Verifying the meaning of P would first involve describing a method whereby it can be checked if a collection of apples is prime. After such a measurement principle is given, we can check for each input instance i_λ, that P's output accurately reflects whether the collection of apples a_λ can or cannot be arranged in more than one row in such a way that there are the same number of objects in every row, but more than one object. The verification would mean a specific empirical decision problem, where we could check instance by instance whether the meaning of P holds.

5.7. The role of the empirical evaluation of meaningfulness

It's important to emphasize the role of empirical evaluation in establishing any computation P as meaningful. According to the original, adopted physicalist definition, meaning is not a static or intrinsic attribute of a formal system L, but a property emerging from both the interpretation and the evaluation of L. This also holds for the translated definition of a meaningful computation, for the following reasons.

A non-halting program does not provide a result that can be assessed for correctness. If halting is not some *a priori* quality of a computation, then it is quite possible that a computation does not halt. Actually, all partial programs are such that they are not guaranteed to produce any output value on some input (Davis 1958, 65)(Crossley et al. 1990, 37). But if it does not, then how would we be able to evaluate the correctness of such a not yet halting physical computation? A core principle of empiricism is that knowledge is limited to what can be observed and understood through the natural world's phenomena. So if a correlation between a computation $P(i_\lambda)$ and the corresponding state of affairs a_λ in the physical world U can't be observed, then it also can't be known. Therefore if meaning is also not some *a priori* quality of a computation, it must be evaluated empirically. But because the correctness of a computation presupposes halting, any empirical evaluation of the correctness of a computation presupposes evaluating its halting behavior. By applying the implicit condition of evaluation resulting from empiricism, a computation under this definition only counts as meaningful only if it is **evaluated** as such. Therefore, any such definition of meaningfulness of computation can not be simply a declaration of meaning or simply a mapping or an interpretation.

Counterfactual completeness of a computation ensures that a computation is not just correct by accident for certain inputs but is inherently capable of correctly solving the decision problem it is designed for, across all possible inputs. This condition means that a meaning carrying computation must be able to consider both possible outcomes. There could also exist false cases of alleged

computations, where the program Q on a range of inputs is predetermined to halt on some value, which then seems to correspond with the actual fact of the world. In any such cases (Example 2), it could happen that there is an apparent correspondence between the results of the computations and the corresponding states of affairs in U. Therefore, in order to exclude such cases from being meaningful, it must be required that a computation could actually produce the other output value than what it has actually produced. Combining this condition with **empiricism** (Bacon 1902, 11), this implies a deeper layer of the verification of meaning. This implicit criterion of the empirical determinability of counterfactual completeness requires not just that *'it could have been otherwise'* but that it determinably could have been so, that this counterfactual possibility must verifiably exist.

Since meaning requires both correct correspondence and halting being evaluated, this implies that a physical object such as P can possess meaning only if other contingently existing physical objects exist which can provide this evaluation. So P can only be meaningful as long as some contingently existing physical process provides the interpretation and both the evaluation to P. Without such interpretation and evaluation, the computations of P, while physically real, would lack the grounding required to be deemed meaningful in relation to the problem UP.

Evaluating halting behavior is a preliminary step for evaluation of both correctness and counterfactual completeness of computations. If we wouldn't be able to evaluate halting behavior, we wouldn't be able to evaluate a large class of computations as meaning. We can examine a couple of critical cases of computations to analyze how **and why we need to evaluate their halting behavior to evaluate them as meaningless.**

5.8. Example 2

Suppose the following Example 2, where $UP2$ represents a physical decision problem with either an answer value of 1 or not known to halt $(?)$ on other values, but it can't output a 0 or a definite 'non-halt' value. The problem $UP2_\lambda$ represents the class of questions *'Does the sun at time t_λ shine?'*. Q is a weather predictor computer program, which predicts if the Sun shines.

Condition A for Program Q

> There exists a class of inputs $\{j_\lambda\}_\lambda$ that Q can process, each representing a distinct time instance for evaluating the presence of sunlight at daytime (E. Szabó 2017, 125). Input j represents a specific point in time or instance, corresponding to the variable t_λ in $UP2$, for evaluating the presence of sunlight during daytime. This corresponds to a class of outputs $\{o_\lambda\}_\lambda$ that Q produces, such that each output indicates whether the Sun shines at the time instance t_λ. Output o is a binary value, 1 or 0. So for each input j_λ, the output o_λ is systematically associated with whether the sun shines at the specific time instance represented by t_λ, reflecting the actual weather conditions.

Condition B for Program Q:

For all possible time instances j_λ (representing specific points in time t_λ):

> If the sun shines at time t_λ (corresponding to input j_λ), then $Q(j_\lambda) = 1$.

If the sun doesn't shine at time t_λ (corresponding to input j_λ), then $Q(j_\lambda) = 0$.

Program Q Execution:

For any given input j_λ:

Q outputs 1 if the prediction or evaluation concludes that the sun shines, reflecting a state of affairs where sunlight is present.
Q does not output any value (loops forever).

Verifying if Q meaningfully represents $UP2$ would require evaluating its computations empirically and checking if they actually correspond to the real world of facts. We can imagine the results of such a systematic process by the following table.

$UP2(i_\lambda)$	$Q(j_\lambda)$
$UP2(i_1)$ *shines*	$Q(i_1) = 1$
$UP2(i_2)$ *shines*	$Q(i_2) = ?$
$UP2(i_3)$ *doesn't shine*	$Q(i_3) = ?$
$UP2(i_4)$ *shines*	$Q(i_4) = 1$
$UP2(i_5)$ *doesn't shine*	$Q(i_5) = ?$

In this scenario, after reviewing these 5 individual cases, the mere existence of any correlation between the computation's predictions and the physical decision problem it's compared to might be considered meaningful. However, this wouldn't reflect a deep understanding of the processes involved, especially if the computation's outcomes are limited or predetermined in a way that excludes genuine variability. In any such case, the computation of Q and the seemingly independent problem constrain the empirical problem to a domain where no discrepancy can be found.

If we wouldn't be able to determine the halting status of computations ─ for example the halting status of Q on $Q(i_2)$, then we would not be able to evaluate Q with respect to $UP2$. Now what follows from this for the meaning of Q? In this case, can we evaluate Q as meaningful or meaningless?

We have to remember that the definition of meaning for computations is such that it requires both the possibility of an occurrence and the possibility of non-occurrence of a correlation, which is explicitly expressed in the definition of meaning. Counterfactual completeness of a computation ensures that a computation is not just correct by accident for certain inputs but is inherently capable of correctly solving the decision problem it is designed for, across all possible inputs. This implicit criterion requires that *'it could have been otherwise'* and determinably so. It could have been that computation Q on input j_λ produces either a 0 or 1 value and this counterfactual possibility must verifiably exist.

If we know for certain that for example on the specific value of (i_2), Q won't ever produce an output value, then we could conclude that Q is meaningless. However, if we can't ascertain this, because what we can observe is that Q hasn't yet halted, then we could not consider it as meaningless. We

can see from this example that the ability to evaluate and determine halting behavior is crucial for evaluating the meaningfulness of programs.

5.9. Halting evaluation with an incorrectly specified evaluator

Evaluating halting behavior is crucial for evaluation of both correctness and counterfactual completeness of computations. This gives us an intermediary but important result to the question *'Can a physical computation H meaningfully represent the halting problem?'* According to this result, H can only represent the halting problem if it itself is evaluated as halting correctly on all instances. And if any evaluation finds that H doesn't halt or doesn't halt correctly, then H is evaluated as meaningless.

It's possible however to **commit a mistake**. It's possible to specify such an evaluation of H by implicitly relying on the abstract description of a Turing machine computation, which contradicts both empiricism and physicalism. Such a mistake can be committed if H is treated on the one hand as a physical evaluator of computational predictions made by physical agents and then on the other hand the computational capabilities of H are treated as if it were an abstract Turing machine.

Such a conflation of positions can lead to the view for example in (Lloyd 2012) that there exist physical computational agents which **fundamentally** can't predict their actions. To see how and why such a mistake is possible, we have to first specify H **incorrectly** and then examine with a couple of example cases where this would create problems. The analysis of this mistake will be crucial to understand how a correctly aligned physicalist approach can handle the halting problem without contradictions.

5.10. Example 3

Suppose the following example, where $UP3$ represents the class of questions: *'Does the computation P_λ halt on some input i_λ?'* (Turing 1936)

Condition A for Computation H:

> There exists a class of input pairs $\{`P'_\lambda, i_\lambda\}_\lambda$ that $H$ can process, each representing a specific program $P_\lambda$ and its specific input $i_\lambda$. Input $(`P'_\lambda, i_\lambda)$ is a pair consisting of a description or Gödel-number based encoding of a specific instance of program P_λ and an input i_λ. Each P_λ represents a unique program or computational process in a standard enumeration of programs (Davis 1958, 56-62), and each i_λ represents a specific input in a standard enumeration of inputs to that program P_λ. To each input of *H* corresponds a class of outputs $\{o_\lambda\}_\lambda$ that H produces, indicating whether each program P_λ halts on its specific input i_λ. Output o_λ is a binary value, 1 or 0, indicating whether the program P_λ halts on input i_λ (1 if $P_\lambda(i_\lambda)$ halts, 0 if $P_\lambda(i_\lambda)$ does not halt). For each input pair $(`P'_\lambda, i_\lambda)$, the output o_λ systematically indicates the halting behavior of P_λ on i_λ, accurately reflecting whether or not P_λ terminates.

Condition B for Computation H:

For all instances P_λ and i_λ:

If $P_\lambda(i_\lambda)$ halts, then $H(\text{'}P\text{'}_\lambda, i_\lambda) = 1$.
If $P_\lambda(i_\lambda)$ does not halt, then $H(\text{'}P\text{'}_\lambda, i_\lambda) = 0$.

For any given input pair $(\text{'}P\text{'}_\lambda, i_\lambda)$:

H evaluates whether P_λ halts on i_λ.
H Outputs 1 if P_λ halts, accurately reflecting a termination state.
H Outputs 0 if P_λ does not halt, accurately reflecting an ongoing computation without termination.

We can consider the following example of a situation by considering it a variant of the undecidability of the halting problem. Here is a variant of how the paradox can be instantiated.

Definition of the anti-diagonal program D, which contradicts the output of $H(\text{'}I\text{'}_\lambda, i_\lambda)$.

If $H(\text{'}I\text{'}_\lambda, i_\lambda) = 1$, then $D(i_\lambda)$ doesn't halt.
If $H(\text{'}I\text{'}_\lambda, i_\lambda) = 0$, then $D(i_\lambda)$ halts.

Program D on input 'D' — contradicts the output of $H(\text{'}I\text{'}_\lambda, i_\lambda)$.

If $H(\text{'}D\text{'}, \text{'}D\text{'}) = 1$, then $D(\text{'}D\text{'})$ doesn't halt.
If $H(\text{'}D\text{'}, \text{'}D\text{'}) = 0$, then $D(\text{'}D\text{'})$ halts.

This results in the following contradiction(s):

If $D(\text{'}D\text{'})$ halts, then $H(\text{'}D\text{'}, \text{'}D\text{'}) = 1$.
If $H(\text{'}D\text{'}, \text{'}D\text{'}) = 1$, then $D(\text{'}D\text{'})$ doesn't halt.

If $D(\text{'}D\text{'})$ doesn't halt, then $H(\text{'}D\text{'}, \text{'}D\text{'}) = 0$.
If $H(\text{'}D\text{'}, \text{'}D\text{'}) = 0$, then $D(\text{'}D\text{'})$ halts.

This can be represented with the following table:

$D(i_\lambda)$	$H(\text{'}D\text{'}, i_\lambda)$
$D(i_1)$ halts	$H(\text{'}D\text{'}, i_1) = 1$
$D(i_2)$ halts	$H(\text{'}D\text{'}, i_2) = 1$
$D(\text{'}D\text{'})$?	$H(\text{'}D\text{'}, \text{'}D\text{'}) = ?$
$D(i_4)$ halts	$H(\text{'}D\text{'}, i_4) = 1$
$D(i_5)$ halts	$H(\text{'}D\text{'}, i_5) = 1$

In this case, is it possible to ever find out if $D(‘D’)$ halts in the physical universe? Also, can we ever **verify** a correspondence between $D(‘D’)$ and some physical state of affairs? Neither the counterfactual completeness criterion, nor the criterion that a computation should first halt in order to be able to be determined correct are met here. And that is because in physical reality there can’t exist states of affairs which are contradictory. Therefore from the point of correspondence, no physical states can correspond to $D(‘D’)$. This means that there exist no seemingly *self-referential* states which *halt on themselves if they don't halt.* Following this line of thought, it follows that D as well as H is meaningless.

From a physical realist point of view, every computational process in the universe has a definite, predetermined outcome: it either halts or does not halt, with no room for indeterminacy. Understood this way, non-halting cases of computation in the physical universe are excluded from being meaningful simply by being physical. If we follow what this implies then in the physical world, there can't exist physical cases of computation where the halting status of a physical process is indeterminate. A computation that does not halt (and thus does not reach a conclusion) is inherently meaningless. In this case, a halting computer, just as any physical program executed on a physical computer which doesn’t halt on a determinate halting state is evaluated as meaningless.

This view is even more strengthened if we consider the first law of thermodynamics, which states that energy in a closed system is conserved, not created or destroyed. This further consideration is rooted in physicalism, as it posits that all phenomena, including computational processes, are subject to and determined by the laws of physics. A Turing-computable process, when implemented physically, must conform to the first law of thermodynamics. It consumes energy to perform computations but does not create or destroy energy. The energy used for computation is transformed from one form to another, adhering to the conservation principle. According to this law, *„concepts such as reversibility and dissipation-free computing are not compatible with Turing's requirements for a computing machine” (Ferry and Porod 2023).* A truly infinite process ─ such as any truly non-halting process ─ would imply perpetual motion or operation without the need for additional energy input beyond what was initially available, challenging the constraints of physical laws. This means that when we're considering a physical computation, then the physical computation should be treated as an eventually halting computation ─ or at least every physical process must be treated as halting from a physical point of view. This would mean that even ff we would accept Turing’s *'logical'* proof that $H(‘D’,’D’)$ can’t produce a correct output value, then it would still follow that both $D(‘D’)$ and H as a physical process, should either halt or run forever, but most likely they will halt in some finite amount of time.

All these empirical and naturalist considerations point to the answer that $D(‘D’)$ is not a meaningful computation and that no meaningful physical computation H can exist either, but also that such lack of meaning must be determinable.

5.11. Can any physical agent determine halting behavior?

If we naively ─ and falsely ─ understand H as a computational but at the same time also a *physical* agent evaluating the halting behavior of $D(‘D’)$, this leads **to a different result**. Suppose we argue based on the findings of the former section that an evaluation of halting behavior is needed to evaluate the halting behavior and therefore also the meaningfulness of a computation. Up until now we haven’t committed any false leap in the argument. But suppose we also accept that the evaluation of halting behavior must be carried out by some physical process in order to decide whether a given physical process is physically meaningful or not. This assumption seems reasonable, if we accept

physicalism. But then, if we also accept the physical Church–Turing thesis in the form that every physical process is Turing-computable (Deutsch 1985). This entails that every physical process, especially the evaluation of halting behavior, should be Turing-computable. This would also entail that the evaluation of meaningfulness of H should also be representable by some computational process.

Based on the arguments laid out in section 3.10, this is not possible. For suppose that such a computer M could determine that $H(\text{'}D\text{'}, \text{'}D\text{'})$ doesn't halt. Then, because D uses H's computation on a given input value as a subroutine, this would also mean that M could determine that $D(\text{'}D\text{'})$ doesn't halt either. And this would contradict Turing's proof. Therefore, no computation even as powerful as M can determine the halting behavior of H. If every physical process must be a computational process and no computational process can ever determine the halting behavior of H, it follows that no meaningful physical process can determine the halting behavior of H.

And this already leads to a contradiction with the former, allegedly *'physicalist'* conclusion that H is not a meaningful computation and that **no physical computation** H can meaningfully represent the halting problem.

Based on such contradictions, and especially based on the fact that Turing's proof is considered *'true'* in some platonistic manner, a lot of thinkers went to the conclusion (Lucas 1961) (Hofstadter 1979) (Penrose 1989) that the human mind can somehow *'see'* this result in an intuitive way, compared to a physical computational process. Therefore it is said that it can also be concluded that the human mind is not entirely physical. If we don't accept any such form of platonism – but we still accept the argumentation above –, then the question remains as to where contradiction can come from.

Suppose that following Chapters 2 and 3, we accept Turing's proof as being part of the set of correct proofs, but we also accept that the set of correct proofs is non-recursively enumerable or non-Turing computable. Therefore, from the physical Church–Turing thesis it follows that Turing's proof might not be enumerable in the set of correct proofs by any physical process. This might give us the idea that it might be Turing's proof which causes the problem. However, as it can be seen following Chapters 2 and 3, any such solution would just deepen the agnosticism towards physical computability. So by accepting that Turing's proof is not a valid, effective or physical proof, what follows is that we have no way to answer the seemingly still open and seemingly still valid question whether $D(\text{'}D\text{'})$ halts or not. But then how would it be possible to solve this contradiction?

5.12. Empirical criteria for halting evaluation

In physically evaluating the halting behavior of H or D, using a powerful abstract halting machine the way it's done in Turing's abstract proof is besides the point. There is an even more stringent requirement. Without specifying what physical measurement process could evaluate a computation as non-halting, the physicalist philosophical principles are not aligned to empiricism. If we don't understand exactly **how** it's possible to carry out halting evaluations, then why should we trust that H or in fact any computation P is meaningful?

If halting or non-halting of any computation is not considered some *a priori* valid quality, then it is a contingent property which must be empirically determinable. It must be either verifiable or falsifiable. The empirical determinability of halting and of non-halting would require an operational definition that outlines how these properties could be tested empirically. Operational definitions provide a bridge between theoretical constructs and measurable, observable events, allowing for the empirical

verification or falsification of the system's assertions. This operational definition would outline the criteria for verifying how P halts on i.

The problem is that if there were no operational criteria at all on how to verify halting behavior empirically, then such a criterion of halting could potentially be considered as either *a priori*, or following Bridgman (Bridgman, 1949), even meaningless. However, the original definition of meaning for physical formal systems (E. Szabó 2017, 8-9) and the explanation of the definition clearly indicates that it can't be a physicalist goal to consider halting of any computation as *a priori*. But if the criterion for halting lacks operational definitions, then it does not explicitly depend on empirical verification and could be understood as depending rather on logical or conceptual validation. This *'logical'* or *'rationalist'* approach to defining the halting would align more with rationalist or platonist perspectives, which hold that certain types of knowledge can be attained through reasoning and logical analysis alone. It contrasts with the spirit of empiricist or pragmatic perspectives, which emphasize the role of sensory experience and the practical consequences of concepts in determining halting. So in order to fill this whole, what's needed is to operationalize the method of verification of halting.

Such an operational definition would require defining the operational methods for the empirical verification of halting behavior. Conditions A and B, as described, do not yet explicitly provide any methods or protocols for empirically testing halting, because they don't yet provide any physically measurable means to check halting behavior. That can't be done by such a general definition of meaning, because each measurement method or protocol must be tailored to the subject in focus. According to (Bridgman 1949, 255), *„definition, or the assigning of meanings, in terms of unique operations is the only safe procedure in physical situations."* Operational definitions should be in terms of unique procedures or sets of operations to measure those physical state of affairs, which are relevant to the measured subject. So in the current case of halting behavior of physical computations, we should set up measurement procedures tailored to this specific property. This is necessary to avoid ambiguity or inconsistency. While Conditions A and B conditions outline the conceptual framework for determining halting computation, the operationalization ─ how to empirically verify by some physical method or process that a given P on i halts ─ would require further specification. A crucial part of defining any empirical verification method for halting behavior would require an observable, measurable halting state.

Setting physical constraints allows us to see how and why the apparent paradox arises if we don't apply such constraints. We can see from the requirement of an empirical measurement process that during the examination of Example 3, by considering H as a computational and a physical agent, we have used **an implicit and wrong method** of verification. We have considered a computation ─ more specifically, an abstract Turing-model of computation ─ to play the *'operational'* role of empirical verification, but **without setting any physical constraints on the measurement on halting behavior.** It seems that we have applied a hidden platonistic element in the argumentations which applied H as a computational but at the same time also a *physical* agent evaluating the halting behavior. Actually, no such H can be specified without contradiction, because it can't be that H is physical on the one hand and yet, on the other hand H operates with unbounded resources.

In its abstract theoretical formulation, the halting problem assumes that the Turing machine has an infinite tape (memory) and can operate for an infinite amount of time. This abstract setup allows for the exploration of the limits of computability without concern for physical constraints. In the abstract halting problem, the naive *'measurement'* done by H is nothing but the following process: simulate P(i) unboundedly, until some definite computational value is reached. For programs that halt, this method would, in principle, allow H to observe the halt and then report it by halting itself and outputting 1. But in the original 'abstract' proof of the halting problem, there is no indication on how H could determine non-halting. Non-halting certainly cannot be decided by a simulation carried out this way.

For a computation to be empirically evaluated for halting, there must be a clearly definable and observable state that indicates completion. So from the point of empirical measurement, it could not be the case that the observed state is undetermined. Because any physical observation process **has to finish within a finite time frame**, building on this would require that the monitoring process would run for a predetermined finite period, after which an assessment is made. If the computation reaches the completion state within this period, it is deemed to have halted; if not, it should be considered non-halting. Without any such specification, the verification of halting would not be empirical. In a correct, physical halting problem setup, halting conditions should be verified empirically, based on the observation of physical computational processes after a finite amount of time. The assumption that $P(i)$ operates within a finite amount of time and has a maximum physical limit size reflects the reality of physical computing systems, which are constrained by memory, processing power, and operational time. But this means that if after some fixed amount of time of measurement, the computation of *H* does not halt, then since non-halting computations can't be considered meaningful, then according to any **empirical evaluation**, *H* can not be meaningful. With this in mind, we can specify a halting evaluator which operates within correct physical constraints.

5.13. Empirical specification of the halting evaluator H

The correct specification of H as an physical verifier for halting behavior of physical computations needs to set finite bounds for the empirical verification. First, specification should set a finite bound for the time spent on the verification process, because any physical observation process has to finish within a finite time frame. Such a finite bound can be set by limiting the number of computational steps that the examined computer has to carry out within which it should either halt or if not, then it is not considered as halting. Second, the specification should also set a finite bound for the maximum size of the physical computers or computer programs under examination. That is because if we consider H as a finite computer consisting of a finite number of physical elements, then it can only represent computations which have equal or less computational *'elements'* than H itself. Such a finite bound on size can be set on the number of internal states of the number of lines of program codes of the machine.

If we consider only a finite, physical meaning evaluator computer which operates in a finite, discrete universe, then the problem domain of the class of questions it represents ─ which consist of physical computer programs P carrying out physical computations (i) ─ is also maximally bounded or strictly finite. Such a finite solution is explained in (Lloyd 2012, 2602-2603). A finite halting problem is one, where the number of computational steps ─ let's call this t ─ is limited in such a way, that if a program P on its own input i doesn't halt within t, then the halting program H computes its output value as $H_t(P,i) = 0$. This means that H_t ─ because it is a finite Turing machine ─ always halts correctly, but it can't itself halt within t. However, in order for a physical solution of the halting problem, some modification of this argument is required. This argument presented here features a finite bound on the internal states as well, which will be important, as this creates a class whose halting status can be determined by some finite, bigger halting evaluator in a bigger, finite class.

Consider a t-finite class of physical computers, which are the class of computers P on input i which halt within t number of computational steps but which also have a t maximum number of internal states. Thus, we can also define a t-finite version of the halting program as follows. If a program P_t

halts on input i within time t then H_t outputs 1, if a program P_t does not halt on input i within time t then H_t outputs 0, **no matter what.** The formal definition of H_t would be as follows:

if $P_t(i)$ halts correctly halts within t computational steps, then $H_t(\text{'}P\text{'}_t, i) = P_t(i)$
if $P_t(i)$ doesn't halt correctly within finite t computational steps, then $H_t(\text{'}P\text{'}_t, i) = 0$

Now it is clear that a physical computer program H_t either halts on 0 or 1, because each such P_t on input i either halts or doesn't halt within t. However, the question is whether H_t *'itself'* as H_t can be in the class of t-finite programs. In order to create a contradiction a similar contradiction to what's in Turing's proof, a t-finite version of the anti-diagonal program D named as D_t can be created as follows:

If $H_t(\text{'}I\text{'}, i) = 1$, then $D_t(i) = 0$
If $H_t(\text{'}I\text{'}, i) = 0$, then $D_t(i) = 1$

That is, $D_t(i) = \neg H_t(\text{'}i\text{'}, i)$

Then the following holds for $D_t(\text{'}D\text{'}_t)$:

if $D_t(\text{'}D\text{'}_t)$ halts correctly within t computational steps, then $H_t(\text{'}D\text{'}_t, \text{'}D\text{'}_t) = D_t(\text{'}D\text{'}_t)$
if $D_t(\text{'}D\text{'}_t)$ doesn't halt correctly within finite t computational steps, then $H_t(\text{'}D\text{'}_t, \text{'}D\text{'}_t) = 0$

This means that if we suppose that $D_t(\text{'}D\text{'}_t)$ halts correctly within t computational steps, then $H_t(\text{'}D\text{'}_t, \text{'}D\text{'}_t)$ must be equal to $D_t(\text{'}D\text{'}_t)$. But then from the definition of H_t and D_t we already have the following:

If $D_t(\text{'}D\text{'}_t)$ halts correctly in t, then $H_t(\text{'}D\text{'}_t, \text{'}D\text{'}_t) = D_t(\text{'}D\text{'})$

Which is a contradiction, because $D_t(\text{'}D\text{'}_t)$ was defined as $= \neg H_t(\text{'}D\text{'}_t, \text{'}D\text{'}_t)$. On the other hand,

if $D_t(\text{'}D\text{'}_t)$ doesn't halt correctly in t, then $H_t(\text{'}D\text{'}_t, \text{'}D\text{'}_t) = 0$
If $H_t(\text{'}D\text{'}_t, \text{'}D\text{'}_t) = 0$, then $D_t(\text{'}D\text{'}_t)$ halts on 1

Which is also a contradiction. It follows from this that **neither H_t nor D_t can be in the class of t-finite programs**. This means that no t-finite halting evaluator H_t can represent or compute the halting behavior or any computation within the same class. It follows that no such H_t , as well as no such D_t is meaningful. **This result is now perfectly in line with the empiricist and naturalist line of argumentation which also arrives at the same conclusion, according to which these are meaningless computations.**

There is another result which can be reached, but which does not contradict these findings. In this given setup, the t-finite class was deliberately constructed in such a way that the size of the class is finite. But if the size is finite, then it's also quite possible that there exists another bigger, but still finite class of computations which can actually represent the smaller class. This could yield a strange result, according to which the halting status of both H_t and D_t is computable by some finite physical computational process. Such a result could be important in that it could explain the apparent contradiction as to why a physical evaluator such as a human is able to evaluate the halting status of computations while a physical computer is apparently not able to do that.

To show that the t-finite class is finite, we can find a general formula for the number of unique computers that halt within t steps, given t states (including one halt state) and we can generalize the conditions and consider the operational specifics of a Turing machine computer under these constraints. Suppose the physical version of the Turing-computer model uses a unary alphabet in our scenario, and we simplify the alphabet to two symbols ($'1'$ for writing and $'0'$ for the blank).

Given these conditions we have:

> t-1 non-halt states and 1 halt state, for a total of t states;
> Unary alphabet, simplified to 2 operations for each step: writing $'1'$ or leaving the cell unchanged;
> 3 possible directions for the tape head to move: left, right, or stay.

Steps 1 to t-1

For the first t-1 steps, the machine can:

> Be in any of the t-1 non-halt states;
> Choose to write $'1'$ or leave the cell unchanged;
> Hove the tape head in one of 3 directions;
> Transition to any of the t states for the next step (but aiming not to halt prematurely).

Each of these steps thus has $[\ (t\text{-}1) \times 2 \times 3 \times t\ ]$ possibilities.

Step t

On the final, t-th step, the machine must halt. It can:

> Be in any of the t-1 non-halt states (as it transitions to the halt state in this step);
> Choose to write $'1'$ or leave the cell unchanged;
> Hove the tape head in one of 3 directions.

For this step, the possibilities are $[\ (t\text{-}1) \times 2 \times 3\ ]$ because the transition is explicitly to the halt state.

The total number of unique machines can then be calculated by multiplying the possibilities for each of the first $t—1$ steps and then multiplying by the possibilities for the final step.

> Total Configurations $= (6t(t\text{-}1))^t/t$

This formula encapsulates the exponential growth in the number of unique configurations as t increases, which reflects both the increase in computational steps and the number of states. It also illustrates the vast diversity of computer programs possible even within such constrained settings. But more importantly, this means that the general formula is a product of the original t, which means that if t is a finite number, then it's possible to simulate by an exhaustive search process to compute the possible t-finite amount of computations of these finite Turing machines and find out if they halt. But this means that the number of unique computers that a halting evaluator program for the class t has to represent is also finite. This means that while no finite computer within the class t can decide its

'own' halting problem, some **other** finite computer H_{t+n} in a class $(t+n)$ – where n is also some finite number – can decide the halting problem of the whole t class. This other program can then make the original programs in the class t comparable for correspondence, because any program in the class t will arrive at a definite output value in the class $(t+n)$.

Suppose we define H_t as H_{t0} and H_{t1} as that next $H(t+n)$, which halts in $(t+n)$ number of computational steps and which has $(t+n)$ number of internal states. H_{t1}, due to its ability to operate within an expanded limit of $(t+n)$ computational steps and $(t+n)$ internal states, can determine the halting problem for any computer P within class t. This way, we can abbreviate the next t where H_t halts on all programs within t as H_{t1}. Then for example the output value of $H_{t0}(D_{t0},D_{t0})$ can also be calculated physically, but not within t. And then we have the following solution.

Here is the definition of H_{t0}:

if $P_{t0}(i)$ halts correctly halts within finite $t0$ steps, then $H_{t0}('P',i) = 1$
if $P_{t0}(i)$ doesn't halt correctly within finite $t0$ steps, then $H_{t0}('P',i) = 0$

Then we can ask the following question this way, if H_{t0} itself was initially conjectured a computer within the class t0:

Does H_{t0} halt correctly within finite steps?

if $H_{t0}('P'_{t0},i)$ halts correctly halts within finite $t0$ steps, then $H_{t1}('H'_{t0},'P',i) = 1$
if $H_{t0}('P'_{t0},i)$ doesn't halt correctly within finite $t0$ steps, then $H_{t1}('H'_{t0},'P',i) = 0$

It can be imagined that it is initially conjectured that the computer H_{t0} is within the class $t0$, but then it turns out that it can only be in the class $t1$. So this means that we would have a hierarchy of t_n and t_n+1 strictly finite computers for which the halting problem would always be computable, meaning that for any strictly finite t_n, its halting problem is always computable by t_n+1. So the question *'does program H_{tn} halt correctly within finite steps?'* is always computable for any H_{tn} in t_n by another computer H_{tn+1} in the bigger class t_n+1.

For the specific $D_{t0}('D'_{t0})$ we would have the following:

if $H_{t0}('D'_{t0},'D'_{t0})$ halts correctly halts within finite $t0$ steps, then $H_{t1}('H'_{t0},'D'_{t0},'D'_{t0}) = 1$
if $H_{t0}('D'_{t0},'D'_{t0})$ doesn't halt correctly within finite $t0$ steps, then $H_{t1}('H'_{t0},'D'_{t0},'D'_{t0}) = 0$

This means the following. **It is H_{t1} and only at least H_{t1} – meaning that only a program which is the bigger finite class of computers $t1$ – which is guaranteed to correctly compute the output value of H_{t0}.** Therefore, only H_{t1} can represent H_{t0} meaningfully.

Suppose that by either some direct calculation or by a finite diagonal argument following Turing's proof we have $H_{t1}('H'_{t0},'D'_{t0},'D'_{t0}) = 0$, meaning that $H_{t0}('D','D')$ doesn't halt correctly within finite $t0$ steps. But we have to remember that this is only according to H_{t1}. If the output value of $H_{t0}('D'_{t0},'D'_{t0})$ is not computable within $t0$, then of course, this means that according to H_{t1},

H_{t0}(‘D’$_{t0}$, ’D’$_{t0}$) doesn't halt in $t0$. If H_{t0}(‘D’$_{t0}$, ’D’$_{t0}$) doesn't halt in $t0$ according to H_{t1}, then because of the specific construction of D_{t0}, D_{t0}(‘D’$_{t0}$) doesn't halt in $t0$ either, but only according to H_{t1}. But this does not mean that H_{t0} can in any way acquire this result and that it itself can output H_{t0}(‘D’$_{t0}$, ’D’$_{t0}$) = 0. So in this case, there is no contradiction or paradox. In any such case, the direct calculation of H_{t0}(‘D’$_{t0}$, ’D’$_{t0}$) not halting, or $t1$-finite version of $Proof\ T$, which asserts the same can still be computed by H_{t1}, but it doesn’t lead to a contradiction.

This has the following consequences. First, it is true for all finite physical processes within class t that they can't be evaluated by a finite physical evaluator H_t belonging to at most class t. Second, it is also true for all finite physical processes within class t that they can be evaluated by another finite physical evaluator H_{t+1} which is part of a bigger finite class of computers $t1$. This shows that a finite version of Argument A can be validly constructed, according to which the following holds: **Given the assumption of physicalism and given a variant of the physical Church–Turing thesis, a finitely specified variant of Turing’s proof shows that there exist finite physical processes within an empirically defined class *t* of physical computers which can't be evaluated by a finite physical evaluator H_t belonging to at most class *t*.**

This result explains the seeming contradiction that a physical evaluator such as a human is able to evaluate the halting status of computations while a physical computer is apparently not able to do that. Actually, this result shows that from the point of view of halting evaluation, there is no special difference in what humans or any kind of physical computational processes can do. In many cases, if the observed or evaluated class is smaller than the class of the evaluator, then even a brute-force verification is possible. In the other cases, when the evaluator is not in the bigger size class, no such evaluation is possible.

5.14. Empirically augmenting H

An important remark has to be made, which is about the empirical constitution of this setup. A Turing machine H by running a mere simulation on a computation P has no way to count the computational steps of other Turing machines such as P. It also can’t by way of simulation measure the number of internal states of other Turing machines. It has to be augmented with physical and empirical semantics to be able to conduct such measurements.

We can define one computational step as executing one instruction, one quadruple of the transition function of a Turing machine. In this context, a quadruple refers to a tuple of four elements that defines a single transition or action the Turing machine takes when in a specific state and reading a specific symbol on its tape. A computational step defined in this way is a full cycle of reading the tape, deciding based on the transition function, acting by writing a new symbol or the same symbol, and moving the tape head. This definition aligns with the conceptual operation of the Turing machine as a simple yet powerful model of computation.

Executing one instruction is then defined semantically as the following action. The Turing machine, in its current state and reading a specific symbol under the tape head, consults its set of quadruples (the transition function) to find the corresponding instruction. Then, according to the found quadruple, the Turing machine writes the specified symbol on the tape, potentially altering what was there before. Then, the tape head moves in the direction specified by the quadruple, which could be left, right, or not at all. Finally, the Turing machine transitions to the next state.

This cycle represents a single, physical, atomic step in the computation of a Turing machine. Augmenting a Turing machine H_{t+1} to count the computational steps of another Turing machine P involves incrementing a physical counter each time H_{t+1} simulates one of P's quadruples during its computation. This however involves a few key modifications to H_{t+1}'s structure and operation.

The simplest way to augment H_{t+1} is to add an extra tape dedicated to counting steps. This tape will be used solely for incrementing a counter each time H_{t+1} simulates a step of P. Each time H_{t+1} completes a simulation of a single computational step of P, it increments the counter on the dedicated tape by way of an additional mechanism. This can be done by encoding numbers in a chosen numeral system (binary, unary, etc.) and incrementing the counter accordingly. H_{t+1}'s state transition function (the rules that govern how H_{t+1} reads symbols, changes states, and moves the tape heads) must be modified to include the action of incrementing the step counter each time a step of P is simulated. This means for every transition that simulates a step of P, there should be an additional step or steps that increment the counter.

To count the internal states of a Turing machine P during its simulation by another Turing machine H_{t+1}, a similar augmentation strategy can be applied. This involves tracking which states P enters throughout the simulation and keeping a count of unique states encountered. Just as with counting computational steps, H_{t+1} would utilize an additional tape or a designated section of an existing tape to keep track of the states that P enters. This would involve encoding each state that P enters in a way that H_{t+1} can recognize and record it without duplication. Each state of P needs to be uniquely identifiable by H_{t+1} so that H_{t+1} can accurately record which states have been encountered. This involves a mapping mechanism or a direct encoding of P's states on the tracking tape. Each time H_{t+1} simulates P entering a new state, H_{t+1} checks the tracking tape to see if this state has been encountered before. If it is a new state, H_{t+1} records it on the tracking tape. The mechanism for checking and recording states must ensure that each unique state is counted only once, even if P enters the same state multiple times during its computation.

However, it does not necessarily follow that just because H can count computational steps and internal states of *t-finite* programs, it has semantics or is more than a symbol-manipulator system. Counting steps and states can be seen as purely syntactic operations - it doesn't require any understanding of the meaning of the program being analyzed, only an ability to recognize and tally up certain syntactic elements (state transitions and unique states encountered). A traditional Turing machine model is already capable of counting and arithmetic. Modifying H to track steps and states of the program it is simulating amounts to having H_{t+1} perform some additional counting operations alongside the core simulation. But it is still operating in a purely mechanical, syntactic fashion.

Semantics implies some level of understanding or attribution of meaning. But H_{t+1} tracking steps and states doesn't mean it understands anything about what the program is actually doing or computing. It's just an augmented syntactic analysis. However, while the augmented H_{t+1} with step and state counting capabilities may not inherently have semantics in the sense of understanding meaning, there are semantic considerations in the design decisions behind limiting H_{t+1} to analyze *t-finite* programs. The choice to focus on *t-finite* programs and to equip H_{t+1} with mechanisms to track steps and states up to the finite bounds is motivated by empirical considerations about the nature of real, physical computing systems. Actual computers have finite memory, finite processing speed, and operate in

finite time. Unbounded computation is a theoretical abstraction that doesn't reflect the constraints of the physical world.

Without an empirical argument for doing so, limiting H_{t+1} to a fixed number of steps and states would be arbitrary. It becomes non-arbitrary especially because we have empirical reasons for it. So the decision to limit H's analysis to *t-finite* programs incorporates some level of empirical semantics - it's not an arbitrary choice but one grounded in our understanding of the physical reality of computation. It reflects the semantic knowledge that real computational processes are finite and bounded.

The specific mechanisms added to H_{t+1} to track steps and states have an empirical semantic basis. They are designed to **measure** properties that have practical relevance in assessing the complexity and feasibility of programs running on actual physical systems. The step count reflects time complexity and the state count reflects space complexity, both of which are important considerations in real-world computing. So while the augmented H_{t+1} may still be operating in a syntactic, mechanical way, the design choices behind its capabilities and limitations incorporate empirical semantics.

The correspondence between physical time and computational steps, and between physical size and number of states, does constitute a form of semantic mapping or correspondence. This way, we are establishing a semantic link between the physical model of computation and the empirical properties of computing systems. We are interpreting the computational steps of the Turing machine as corresponding to the passage of physical time, and the number of states as corresponding to the physical size or space complexity of the system. Therefore, the correspondence between time and computational steps, and between size and number of states, is a semantic correspondence. It's a way of imbuing the H_{t+1} with physical meaning based on our empirical understanding of computation in the real world. By designing the augmented H_{t+1} to track steps and states, and limiting its analysis to t-finite programs, we are effectively building these semantic correspondences into its operation.

What this augmentation means is that this enhanced Turing machine H_{t+1} now has a predefined, operationally defined meaning to physically measure both the number of computational steps and the number of internal states of other Turing machines. H_{t+1} with these capabilities, it becomes not just a simulator of P's computation but also an analyzer that can provide insights into the computational complexity of P's operation. But the important thing here is that this means that H_{t+1} now has an actual, physical semantics (Wegner 1997) defined for counting steps and internal states. This quantification is not merely abstract but is physically represented within the computational setup of H_{t+1}. But defined this way, H_{t+1} is not a simple symbol-manipulator system and also can't operate in an abstract way anymore. It will be bound to physical properties. Every tape movement, symbol write/read operation, and state transition consumes energy and takes time, so H_{t+1}'s operations will be subject to physical constraints and laws, including the laws of thermodynamics.

This has the following consequence. While a mere symbol-manipulator (Newell 1976) H can't be known to solve the problem whether such allegedly self-referential, abstract computations such as $D(\text{'}D\text{'})$ or $H(\text{'}H\text{'})$ halt or not and therefore can't in general decide the correctness or meaningfulness of computations, the physically defined H_{t+1} at least can decide the correctness or meaningfulness of a finite t class of computations.

5.15. Does the physical Church–Turing thesis tacitly suppose platonism?

The argument applying the physical Church–Turing thesis in connection with Turing's proof of the uncomputability of halting problem originally given in a form that promotes physicalism, so if it were to suppose platonism in any form, then it would undermine the goal of the argument. Up until this point, we have been focusing on how one part of this argument - utilizing Turing's proof - can be interpreted as an abstract, therefore platonistic element. If however, we're applying a correct, empirically augmented, *t-finite* variant of Turing's original proof, we can get rid of this platonistic element.

There is however a second objection of platonism which can be given against the core argument. It could also be argued that the physical Church–Turing thesis itself contains an element of platonism. This can be done in the following ways.

If the core assumption of the PCTT that *'every finite physical system can be simulated by a finitely operating computer'* is understood in such a way that the representing or simulating computer is imagined as an abstract entity, which doesn't operate in the physical universe, then of course the PCTT itself presupposes the existence of a type non-physical computation which would undermine the physicalist nature of the original argument. Against such views, we can point to the physico-formalist philosophy of mathematics, as outlined by László E. Szabó (E. Szabó 2017, 3-5), which is rooted in the integration of three foundational philosophical premises: physicalism, empiricism, and formalism. This argues that all logical and mathematical entities are not abstract but instead physically instantiated in the world and that all logical and mathematical truths are contingent on physical reality. The core of the argument for the purely physical interpretability of the representing formal system can be given in three steps.

Step I begins by asserting that formal systems can be represented physically. We can use the example of a computer with a CD (E. Szabó 2017, 3) that contains a program which lists the theorems of a formal system. This setup illustrates that the formal system (typically viewed as an abstract entity) is actually instantiated in a physical object – the computer. The key assertion here is that the operations and outputs of this computer (e.g., displaying theorems) are physical manifestations of the formal system, demonstrating that the system's properties and operations are grounded in the physical setup of the computer. In Step II (E. Szabó 2017, 4), we can argue that the consideration of any formal system inherently involves imagining it as being represented physically. This aligns with the view of computation, according to which abstract mathematical concepts are only accessible through physical means, such as computers or human brains. The implication is that these abstract systems are concretely realized and their truths are physically instantiated. Step III (E. Szabó 2017, 4-5) concerns the notion of representation itself, arguing that there is nothing beyond the physical manifestations of formal systems: there are no abstract entities being represented. Instead, what is often thought of as an "abstract formal system" is, in reality, just another instance of a physical system. The argument for Step III starts by acknowledging that particular formal systems such as L_1, L_2, ... , L_n are physically instantiated – each system might be represented on different physical entities, such as various computers, pieces of paper, or configurations of other physical materials. When abstracting from these systems, the focus shifts to identifying what is common among them – essentially, what makes them instances of the same type of formal system despite their physical differences. This could include shared logical structures, operations, or rules.

Then, to describe these common features and make meaningful statements about the similarities or structural isomorphisms among L_1, L_2, ... , L_n, a higher-level physical theory (M, S) is necessary. This theory itself is a formal system that describes and organizes the relationships between the lower-level systems. The key element is to realize that the theory (M, S) – while it may sound

abstract because it deals with the relationships and structures spanning multiple physical systems — is itself a physical system. This means that (M, S) can also be instantiated physically (e.g., as a set of equations on a blackboard, a computer model, or in the neural networks of a human brain). It does not follow from anything that it should exist in a non-physical, abstract realm; rather, it exists as another layer of physical instantiation that we use to organize and understand the lower-level physical systems. Thus, when we perform abstraction in mathematics or logic under this view, we are not accessing a non-physical abstract realm but are simply restructuring our understanding of the physical representations and their interrelations within the physical universe. Abstraction, then, is a process of organizing physical information in a physically instantiated system (M, S).

If we interpret the application of the physical Church–Turing thesis this way, namely that we think of the representing formal system also as a physical entity, then we can maintain the integrity of the physicalism of our original argument. Spelled out this way, the PCTT is a benign assertion. It says nothing more than that the representing formal system used doesn't exceed the derivational capacities of a Turing-machine.

It could also be argued that the notion of representation or of meaning necessarily involves conceptual elements. Against such a view, an argument presented by E. Szabó can establish that the semantics of a formal system — how its symbols and formulas correspond to the physical world — does not involve abstract or purely conceptual elements. Instead, it's grounded in physical processes and causal relationships.

The starting point to consider is that semantics is not just an arbitrary link between linguistic elements (symbols, formulas) of a theory (L) and the physical world (U), but rather, it must be based on a real, empirically observable correlation between these two. Condition B specifies that for a formula in L to meaningfully represent a state of affairs in U, there must be a consistent and observable physical relationship between these representations and real-world phenomena.

The argument incorporates the thesis of the causal closeness of the physical world, which asserts that all physical phenomena are interconnected through causal relationships. This means that the relations defined by semantics within a formal system must also be causal: they must be explainable through direct or indirect physical interactions between the elements of L and the phenomena in U. Furthermore, if two events are correlated, there must be a common cause that explains this correlation. In the context of semantics, this implies that the correspondence between L and U is not coincidental but is causally grounded. There must exist a common cause or a set of causes in the physical world - which we can imagine to be physically instantiated as a physical process of learning - that makes the semantic assignments meaningful.

Based on these arguments, we can conclude that all knowledge, including the semantic understanding of formal systems, arises from empirical experience. The semantics of a formal system are developed and validated through interaction with the physical world, observing how changes in U correlate with representations in L. This aligns strongly with empiricist philosophy, which holds that all knowledge comes from sensory experience.

It can also be argued that the physical Church–Turing itself is not an empirical theory, which would also entail that the core argument is not entirely a physicalist one. Against this view, it can be pointed out that David Deutsch's argument itself distinguishes the Church–Turing hypothesis from a purely theoretical conjecture as a physical principle which is empirical and potentially falsifiable (Deutsch 1985, 4-5). Following Karl Popper's philosophy of science, Deutsch asserts that for any principle to be considered scientific, it must be falsifiable. This means there should be conceivable observations or experiments that could potentially contradict the principle. The PCTT, according to Deutsch, fits this

criterion because it could be refuted by demonstrating a physical system that performs computations which cannot be simulated by any universal computing machine. Deutsch acknowledges that some principles, like the principle of conservation of energy or the Church–Turing principle itself, cannot be directly tested or observed in isolation. Instead, their falsifiability comes through indirect means. For instance, if a new physical theory predicts phenomena that violate the Church–Turing principle, and these predictions are experimentally verified, then the principle would be falsified indirectly. This would of course challenge the assumption that all physically realizable computational processes must conform to the traditional, Turing-computable algorithms and would allow the possibility that the physical universe might potentially allow for processes that compute functions beyond those classically considered computable. This would open up the possibility that new types of computing machines might be built that exploit these non-classical functions, providing direct empirical tests of the physical Church–Turing thesis. Such a discovery would not only falsify the principle but also fundamentally change our understanding of physics and computation. While this would challenge the current boundaries of computational theory, this would nevertheless be possible in an empirical interpretation of the PCTT.

5.16. Summary and conclusions

We have defined two critical conditions (A and B) that establish a systematic physical and empirical basis for computations to meaningfully represent or predict physical states of affairs. Condition A requires a one-to-one correspondence between computational processes and physical realities, while Condition B demands that these computations accurately reflect the occurrence or non-occurrence of these realities through verifiable outcomes. Meaningful computations must halt with correct outputs and in order for a computation to be meaningful its halting behavior must be empirically verifiable – otherwise it would not be possible to decide if the computation in question corresponds to reality or not. This implicit condition relies upon the practical verification of computations against the physical reality. It ensures that predictions are not only theoretically sound but also empirically grounded.
In the following sections it is shown with explicit examples not only that the physical processes D and H are not meaningful from a common sense physicalist view but also that evaluating halting behavior is crucial for evaluating meaningfulness. Neither D, nor H cannot meaningfully represent the halting problem because they either lead to paradoxes (as shown in the setup with D contradicting H's output) or fail to align with physicalist constraints. D, as an anti-diagonal program, and H, as its evaluator, are shown to be engaged in tasks that inherently produce contradictions under physical law and logical consistency.

For a computation to be considered meaningful, it must not only halt, but its halting behavior must be empirically verifiable. This requirement aligns with the principles of empiricism, which emphasize observable and verifiable phenomena as the basis for knowledge. Also, from a physical realist perspective, every computational process is determined to either halt or not, with no ambiguity allowed. Computations that do not definitively halt are, by this view, inherently meaningless because they cannot produce conclusive, verifiable outcomes. All empirical and naturalist considerations point to the answer that $D(\text{‘}D\text{’})$ is not a meaningful computation and that no meaningful physical computation H can exist either.

Evaluating halting behavior is necessary to assess the meaningfulness of computations. This builds on the conclusion from Section 5.10 that meaningful computations must be empirically verifiable, particularly in their halting behavior. By accepting physicalism, it follows that any such evaluation of meaning must be carried out by a physical process.

Suppose we argue based on the findings of the former section that an evaluation of halting behavior is needed to evaluate the halting behavior and therefore also the meaningfulness of a computation. Suppose we also accept that the evaluation of halting behavior must be carried out by some physical process. Then the PCTT entails that every physical process, especially the evaluation of halting behavior, should be Turing-computable, which would also entail that the evaluation of meaningfulness of H should also be representable by some computational process.

This however is not possible. For suppose that such a computer M could determine that $H(\text{'}D\text{'}, \text{'}D\text{'})$ doesn't halt. Then, because D uses H's computation on a given input value as a subroutine, this would also mean that M could determine that $D(\text{'}D\text{'})$ doesn't halt either. And this would contradict Turing's proof. Therefore, no computation even as powerful as M can determine the halting behavior of H.

If every physical process must be computable and no computable process can determine the halting behavior of H, it follows that no physical process can effectively determine this either. This presents a paradox where physical processes, which are supposed to be capable of all computable actions, cannot evaluate something as fundamental as halting behavior.

And this is a significant problem, because the former section, 5.10 argues that neither D and H are meaningful from a common sense physicalist view and that evaluating halting behavior would be crucial for evaluating meaningfulness. The inability to determine halting behavior undermines the entire framework for evaluating meaningfulness in physical computational processes.

The contradiction in Argument A arises from the problem of predictability in deterministic physical systems and the limits imposed by the halting problem. If a computational prediction is unbounded, then it operates with endless resources and doesn't pose any measurable condition for halting and non-halting. A solution to this problem can be given through the notion of bounded predictions, where the halting condition for these predictions is operationally defined.

Operational definitions specify the exact procedures or actions required to measure a concept. For bounded predictions, operational definitions provide clear observable, measurable criteria for what constitutes a computation halting, a computational prediction being correct, or a physical state being accurately represented by a computational process. To operationalize the halting condition, the halting evaluator H needs to be augmented with the capability to count computational steps and monitor internal states. Setting finite bounds for the time spent on verifying the halting behavior of computations are required for H. This means that for a computation to be considered as halting, it must complete within a predefined finite number of computational steps. This introduces an empirically measurable criterion for halting, whether or not a computation reaches a conclusion within these finite bounds.

A *t-finite* computational process is a computation that is bounded by a finite number of steps and states, denoted by *t*. A computation or a prediction that is about the halting behavior of another computation within the same *t-finite* class is inherently unpredictable. This is due to the constraints imposed by a finite variant of the diagonal argument of Turing's proof, where a computation cannot predict its own halting behavior or that of another computation within the same bounded conditions without running into paradoxes or contradictions. This unpredictability within a *t-finite* class is a very real consequence which can actually be used to explain why a finite physical agent can feel free or why some sensory information is unpredictable for a physical computational agent, which can give a realistic explanation of qualia. This also allows us to validly state a finite version of Argument A, according to which the following holds: Given the assumption of physicalism and given a variant of the physical Church–Turing thesis, a finitely specified variant of Turing's proof shows that there exist finite

physical processes within an empirically defined class t of physical computers which can't be evaluated by a finite physical evaluator H_t belonging to at most class t.

However, there is a critical distinction when it comes to predictions made by a *t+1-finite* physical system about a *t-finite* system. A computational process that operates within a slightly larger bound ($t+1$) — meaning that it has access to more computational steps or states than the *t-finite* processes it is evaluating — can predict the halting behavior of those *t-finite* processes without contradiction. The *t+1-finite* system has enough computational resources to evaluate the *t-finite* systems' outcomes within the constraints of physical laws and empirical verification.

This distinction resolves the core problem of this analysis, which is agnosticism, i.e. the general predictability dilemma by suggesting that while computations are bound by certain limitations (physical laws, finite resources, and empirical verifiability), they are not universally unpredictable. The above results based on correct empirical specifications if Argument A mean that by requiring a measurable empirical criteria of halting, the problem of unpredictable physical states of affairs disappears. This has the consequence that all computations under empirical criteria of halting can be evaluated for correctness and meaningfulness. Predictions and meaning evaluations about the behavior of simpler or more constrained systems are possible from the standpoint of a slightly more complex or less constrained evaluator. This implies that computational predictability and meaningfulness are not absolute concepts but are relative to the computational resources and bounds of the observer or the evaluating system. This conclusion suggests a hierarchy of computational predictability, where systems can make meaningful and accurate predictions about the behavior of less complex systems. This aligns with physicalist principles by grounding computational processes in the physical universe's laws and constraints and adheres to empiricism by requiring that these predictions should be empirically verifiable.

6. Overview of arguments for free will from unpredictability

6.1. Karl Popper's arguments on predictability and indeterminism

The idea of physical systems unable to predict themselves goes back to Karl Popper. Being an indeterminist, Popper's argument against determinism, and within it what Popper called scientific determinism, was first published in the article *'Indeterminism in Quantum Physics and in Classical Physics'* (Popper 1950) and later clarified in the book *Open Universe* (Popper 1982). Scientific determinism is the view that (in the case of a Hempel-Popper type deductive-nomological model (Hempel and Oppenheim 1948), given perfectly accurate knowledge of the initial conditions and accurate knowledge of the physical theory that accurately describes the workings of the universe) a Laplace demon would be able to predict with arbitrary accuracy any future event (Woodward 2021). According to Popper, the assumption of this idea is an implicit idea in Newtonian physics, and an implicit idea that he does not believe to be true. The idea of physical systems that are incapable of predicting themselves was born as a critique of this implicit idea. For if it is correct, then if the universe as a whole is taken as a physical system, then the universe or any part of it is incapable of predicting future states accurately. Popper outlines three somewhat separate arguments to prove this. Below I reconstruct Popper's original arguments based on the article (Popper 1950).

The first, the basic concept of the Tristram Shandy argument (cf. Popper 1950) is based on the following story, which has been called a paradox. Tristram Shandy is a writer writing his own life story. Unfortunately, however, it takes him longer to write the story of each day than it took him to live the events, so that not only does he never finish the book, but he is constantly moving further and further away from a solution. Consider $t_{1,}$ $t_{2,}$ and t_3 as follows. t_1 = start date of events to be written, t_2 = end date of events to be written and start date of writing, t_3 = end date of writing. The writing is done in the time interval between t_2 and t_3.

To generate the prediction problem, suppose that we are not trying to describe the past, i.e. we are not trying to describe the events of a finite day, we are not trying to describe a finite number of time intervals t_1 and t_2, but we are trying to describe the future. Then the description of the situation changes to the following. Suppose further that the interval between t_1 and t_2, as well as between t_2 and t_3, will always have a minimum length, simply because it always takes all physical entities time to perform any action, including writing.

t_1 = The time the information was obtained, the time the information was processed;
t_2 = The time at which information processing ends, and the time at which information processing starts between t_1 and t_2 about the state of itself, which has changed in the meantime;
t_3 = The time at which t_1 to t_2 starts to process information about its own changed state;
t_3 = The originally scheduled end date of the prediction. In fact, this is the starting time for processing information about the state of itself between t_2 and t_3, which has changed in the meantime. At this time, the system has not yet finished processing the information.

It is clear that even if in this case the story is modified so that the time interval between t_2 and t_3 becomes shorter and shorter, the physical system C performing the prediction will never be able to predict its future state, since the change of state during the interval between t_2 and t_3 must be included in the initial conditions of the prediction, and this interval will always have a minimum length,

since C is a real physical machine. If this reasoning is correct, then no physical machine can predict its own future states, precisely because to do so it would have to take into account the non-zero length of the states that have changed during the time interval during the prediction.

Popper suggests two potential objections to this argument (cf. Popper 1950). The first is that it is conceivable that at time t_2 the machine is still capable of predicting itself accurately at time t_3, in some indirect way. If, for example, machine C gives not only a description of itself but also a description of itself plus the environment, while at the same time accurately predicting itself at time t_3, it may easily turn out afterwards that C has indeed made a correct prediction. A defense against this objection is to require that the time taken to process the information about the changed state of itself between t_1 and t_2, received at time t_2 , is longer than the time taken between t_1 and t_2. This is possible if we take into account that for a physical machine, the processing of information will in any case take longer than the time it takes to read the information, which may imply a different processing process translated into another internal language.

The second objection is that there could easily exist a $C+$ machine that has no problem predicting the future states of C. The question then arises, why is it so difficult to predict the future states of C? Would it not be easier to ask a machine $C+$ about C's states and then accept the answer as a prediction? The answer to this counter–argument is that $C+$ can only perfectly predict the states of C as long as $C+$ is completely independent of C. However, when both $C+$ and C receive information about C, then $C+$ is bound to predict about itself, since C will have part of $C+$'s information, so in this case a self-prediction situation arises for $C+$ in the same way as for C. As soon as interaction occurs between the two systems in the form of information exchange, the two machines will be considered as one, and together the original predictability conclusion holds.

Popper's second, Gödelian argument can be summarized as follows (cf. Popper 1950). Popper concludes that for every computer system there will be a true inference that the system will not be able to compute. Based on Gödel's First Incompleteness Theorem (Crossley et al. 1990, 45), Popper assumes that for any sufficiently complicated formal system S_1 (or physical system realizing a formal system), there exists a well–formed formula g, or Gödel formula, which cannot be derived in the system S_1, nor can its negation be derived in the formal system. According to Popper, if this g *is* such a Gödel formula for S_1, then the true statement h that *'g is undecidable in S_1'* will also be undecidable in S_1 . Popper then argues that it is easy to formulate such a statement h, which will then be a statement about the future physical capabilities of C in response to a particular physical stimulus. As soon as such a question is posed to C in such a way as to stimulate C's system to a YES or NO response, C will be unable to decide on it in such a way that the predication actually retains its predictive character. At time t_1, C can be asked a question q_1 about his own future state at time t_2. The question q_1 could be *'if you are asked a question q_2 at time t_2, will you be able to answer YES or NO to this question?'* Popper's conclusion is that if the above problem statement is correct, then physical system C cannot answer the question q_1. Popper also says that physical system C could answer q_1 in such a way that after a certain time, say time t_2, the answer to q_1 is simply added to C's memory. Then, of course, the question will be answered, but on the one hand this will not be predictive, and on the other hand C's own state will change, so that it will in fact become a physical system C'.

What if C is faced with a time-limited decision? Such a decision is more in line with reality, where a physical structure, in the absence of time or other resources, has to answer questions that need to be

decided. A time-constrained question posed to C can be formulated as follows t_1 in a time instant. Suppose that C_1 and C_2 represent different states of the same formal system at times t_1 and t_2. So the question for C is *'If you are asked the Gödelian question q_2 at time t_2, will you be able to answer it before time t_3 ?'* For C_2 the question q_2 would certainly be a Gödelian question, but C_1 cannot then know that q_2 is too difficult, since by the general argument above, C cannot know this at time t_1.

Popper's third argument is the Oedipus argument (cf. Popper 1950). This is an argument against a false thesis. The thesis is based on an objection to the Tristram Shandy argument, a modification of it. The objection reads as follows: Even though the knowledge of machine C about all its physical states at any given time is incomplete, there may still exist a machine C+ which has complete information about C. Of course, this is true, but, as explained earlier, C+ cannot communicate this information to C, otherwise the two systems would behave as one system and would be unable to make accurate predictions. However, the Oedipus argument may provide a solution to this problem. According to this argument, there may be a clever $C{+}{+}$ machine that can anticipate this problem and, based on this, communicate to C exactly as much and as little information as C needs to be in exactly the same state at time t_2 , i.e. at the end of the prediction, as C predicted at time t_1. Except that, according to Popper, this information that C receives from $C{+}{+}$ can in no way be perfect, and at the same time correct, information about itself, C, since this information would also have to be translated by C from an external language into its own internal language, along the lines of the first argument, and this cannot be completed by time t_2.

As a brief conclusion, it is worth noting that Popper's first argument makes realistic assumptions, but in any case additional assumptions. First, he introduces the lemma that there must be an external and an internal language. This is necessary to ensure that translation between the two languages takes computational time in any case. This assumption seems *prima facie* realistic for any physical machine, but there is little to support it other than to make the argument workable. Indeed, we do not know whether the translation of any information cannot take the same time as the reading of the information. And Popper's Gödel argument makes a rather strong presupposition. That is, there exists a true statement h that *'g is undecidable in S_1 '* will also be undecidable in S_1. In this case, it is questionable, or at least subject to investigation, whether statement h does indeed express a true statement.

6.2. Is Popper's argument that h *is a true statement'* correct?

In order to clarify Popper*'s claim (cf. Popper 1950), it is necessary to examine whether the statement* h*, i.e. '*g is undecidable in S_1', is also a *true* statement which is undecidable in S_1. When can we say that such a meta-arithmetic statement is *true*? The definition of truth and meaning is the following, based on László E. Szabó (E. Szabó 2017, 8–9.):

"... one is entitled to say that a formula A *represents or means a state of affairs* a *in* U*, if the following two conditions are met*

a. *There exists a family* $\{A_\lambda\}_\lambda$ *of formulas in* L*, and a family* $\{a_\lambda\}_\lambda$ *of state of affairs in* U *such that* $A = A_{\lambda 0}$ *and a* $= a_{\lambda 0}$ *for some* λ_0.
b. *For all* λ.

if a_λ *is the case in U,* *then* $\Sigma_L \vdash \boldsymbol{A_\lambda}$,
if a_λ *is not the case in U,* *then* $\Sigma_L \vdash \neg \boldsymbol{A_\lambda}$."

As for the meta-arithmetic propositions that we can represent in Peano arithmetic – the idea of which representation we can get from the proof of Gödel's first incompleteness theorem – we can say the following:

"Let $Pr(x, y)$ *denote the meta-arithmetic fact that the formula-sequence of Gödel number* x *constitutes a proof of the formula of Gödel number* y*. Accordiing to Gödel's construction, every such meta-arithmetic fact* $Pr(x, y)$ *is represented with a PA formula* $R(x, y)$ *in* PA *if the following condition is met. For all Gödel numbers* (x, y)*, it holds that*

if $Pr(x, y)$ *is the case,* *then* $\Sigma_{PA} \vdash \boldsymbol{R(x, y)}$
if $Pr(x, y)$ *is not the case,* *then* $\Sigma_{PA} \vdash \neg \boldsymbol{R(x, y)}$."

The above criterion of meaning becomes a criterion of truth if we also take into account the following statement, which is one of the central claims of László E. Szabó's article (E.Szabó 2017):

"It is important to realize that condition (b) is nothing but a necessary and sufficient condition for this physical theory (L, S) to be true. That is, the two concepts of meaning and truth are completely intertwined."

Consider the formula $\neg \exists x Pr(x, y, y)$ in arithmetic, where the formula sequence of Gödel number x is a proof of the formula obtained from the formula of Gödel number y, having one free variable, by replacing the variable with number (individuum constant) y. Let the Gödel number of this formula be g. The next sentence is called a Gödel's sentence: $\neg \exists x Pr(x, g, g)$. This is exactly the G statement for which the meta-arithmetic statement *'*G *is undecidable in* Σ_{PA} *'* is the corresponding meta-arithmetic statement. However, since this statement is undecidable in Σ_{PA}, it can neither have meaning nor be true by the above criterion. This evaluation holds naturally if we accept this formal (Gödelian) theory of meaning. On this basis, however, it seems that Popper's argument is flawed.

If Popper's reasoning were to be considered correct, it would be possible to do so, if the meta-arithmetic facts $Pr(x, y)$ were not represented in Σ_{PA} but in another (meta) formal system Σ_{PA+1}. The question is how such a system should be conceived, e.g. what is the ontological status of this formal system. The problem is that until this is explained, the question of the truth of proposition h is unclear.

6.3. D. M. MacKay's arguments on free will arising from unpredictability

MacKay (MacKay 1967) put the question of predictability in terms of the problem of free will. He criticized the mechanistic claim that a completely mechanistic theory of brain processes, together with the assumption of determinism, leads to the conclusion that human actions are necessary and inevitable, and that free will is therefore merely an illusion. MacKay posed the question as follows. Let's assume that the functioning of the human brain is as mechanistic as a clockwork, and is completely understandable by a deterministic analysis. What does this imply about the capacity for free will? According to MacKay's argument, nothing that would result in free will being an illusion. MacKay's argument is essentially down to one central point: that any information acquisition is possible only and exclusively through physical changes in the brain. And this physical change is not irreducible, especially if one considers that one's thoughts, and any small change in them, are accompanied by exactly the same changes in the physical brain. It follows that, even if the human brain were a clockwork, it would not be accurate for a human agent to describe this brain state, or even to predict a future brain state. The act of obtaining a description, the act of acquiring information, leads to a direct change in the human being, and a prediction can be regarded as precisely such an act of acquiring information. And if brain states change during the information acquisition or the resulting prediction, the basis of the prediction and the initial conditions of the prediction also change. As a consequence, the previous prediction becomes inaccurate, and this kind of inaccuracy cannot be eliminated by the nature of things.

It is easy, of course, for a description to be accurate up to the point at which one acquires it ─ so that one's actions can be perfectly predictable from an external point of view. Just as it is possible to communicate information to a human agent which will put him in exactly the state which he himself will predict about himself; in which case, however, the information itself would be false if he did not believe it. The consequence of this is that the agent has no reason not to believe it (cf. MacKay 1967).

The implication of the above argument is that this brain description or prediction is not at all independent, objective, but its truth is entirely dependent on the agent or person making the description or prediction. As a result, the epistemological status of the brain state description is not like that of scientific theories, it is not independent of the cognizing subject. According to MacKay, it follows from the above that the mechanistic thesis that what one believes and thinks is strictly reflected in brain states is not true either. The internal discourse in the mind ─ mind-talk ─ and the correlated physical discourse in the brain ─ brain-talk ─ do not translate perfectly into each other.

Let us consider the following situation, which MacKay also argues is a corollary of the above line of thought. Suppose that at 7 pm I have to make a decision about whether or not to take the 7:15 train. Before making this decision, there is no definitive answer as to what the prediction is as to what I will decide. There are two possible descriptions of the world: one ─ G ─ in which I get on the train, and the other ─ W ─ in which I do not get on the train. If the above reasoning is correct, then neither answer is inconsistent with my current knowledge. Add to this the fact that to an outside observer my actions are determined, and what MacKay calls the principle of logical relativity follows: truths are not

truths that will apply equally to all persons. Some of them will be truths that are true for one person and false for another.

On the question of free will, MacKay's conclusion (cf. MacKay 1967) from the above is that since neither G nor W follows from an accurate physical description of brain states, but is also not inconsistent with the knowledge of the human subject, the decision to make a choice can be regarded as a free act. If we accept this, then the conclusion is that a fully mechanistic theory of brain function is not inconsistent with the fact that our decisions can be free.

6.4. Adolf Grünbaum's critical comment on Mackay's arguments

Grünbaum's (Grünbaum 1971) critical comments nuance MacKay's claims. According to Grünbaum, a central element of MacKay's theory is that there is no objective specification for the subject performing the predication that will result in the subject's actions being specified in an objective way, since the subjective agent's decision also fundamentally affects the objective specification. However, Grünbaum argues that this thesis does not necessarily imply that the options G and W described by MacKay have real causal implications in terms of logical predictability. The key to understanding Grünbaum's position is to understand the notion of multiple realizability (Bickle 2020). For if it is possible, as a good number of functionalist philosophers of mind think, that the same state of mind can be realized by several different physical systems, then MacKay's argument ─ that the brain as a physical entity is incapable of predication on its own ─ does not necessarily imply that the states of mind realized by two different brain states G and W must also necessarily be two different states of mind.

„Note that what is at issue is the autopredictability of those features of the brain state which are causally relevant to choice behavior. We know that on the purely psychological level, choices are causally influenced by beliefs pertaining to the consequences of our actions and to some other related matters. But neither this fact nor MacKay's general assumption $S_{psi} = f(S_p)$ *rule out the following possibility: Two kinds of brain state which differ only with respect to the modifying effects corresponding to the agent's believing or not believing a statement as to the physical state of his brain at 7 P.M. may well each be causally coupled to the decision outcome* G*, instead of being respectively associated with the different outcomes* G *and* W *among which the choice was made!"* (Grünbaum 1971)

That is, following the above line of reasoning, it is not at all certain that the options G and W that exist for the agent actually have real causal effects. There is nothing to say that two different brain states do not implement a single mental causation G that is identifiable with both brain state outcomes G and W.

6.5. Ontological and epistemic determinism, predictability

Another remark by Grünbaum is that we must distinguish between the notions of epistemic predictability and ontological predictability. By the former we mean predictability explicitly within the system, by the latter we mean predictability in general. In the first case, predictability means that, given a perfectly accurate knowledge of the initial conditions and a precise knowledge of the physical theory that accurately describes the operation of the universe, we are able to predict with arbitrary accuracy any future event (see Grünbaum 1971, 12).

The idea of epistemic determinism in its most famous form was introduced by Laplace in his *'Essai philosophique sur les probabilités'* (Laplace, 1814, 2009) with the hypothesis of a being known as Laplace's demon.

„... the present state of the universe is regarded as the cause of the previous state and the cause of the state to come. An intellect which knew all the forces of nature and the position of its constituent units at a given moment in time, and which was sufficiently profound to analyze these data, could put the motion of the largest body and the lightest atom in the world into a single formula. Nothing would be uncertain before him; he would see the future as he sees the present, as well as the past. The human intellect, with the perfection achieved in astronomy, could serve as a vague skeleton of such an intellect." (Laplace 2009, 35)

The same idea is reflected in Popper's definition of scientific determinism:

„(Scientific determinism) is the doctrine that the state of a closed physical system at any future moment can be predicted with arbitrary accuracy, even from within the system, by deriving our predictions from scientific theories, given the initial conditions, the necessary accuracy of which can always be calculated [...] if we know what the prediction task is. The demon, like a man of science, does not need to know the initial conditions with absolute mathematical precision; like the scientist, he must be satisfied with finite precision." (Popper 1982, 8)

In contrast, the idea of ontological determinism is independent of predictability. Nuel Belnap describes the objective modality by which the idea of ontological determinism can be simply captured as follows:

„At any given moment in the history of the world, there are many possibilities for how things might continue. Before we flip a coin, two possible things can happen: heads or tails. These two possibilities exist not only in an epistemic sense, but in reality." (Belnap 1992, 5)

If, then, we think that the world is truly deterministic, and not just in the sense of predictability or computability or scientific determinism, then we are effectively saying: let W be the class of all physically possible worlds. Then for two possible worlds it is true that $w1, w2 \in W$. If $w1 = w2$ is true at time t, i.e., if all properties of the two worlds are completely identical at that time, then for all $t < t'$ it is true that $w1 = w2$. All possible worlds are the same in this case. The same idea is expressed in Karl Popper's definition of metaphysical determinism:

„The metaphysical doctrine of determinism simply asserts that all events in this world are fixed, or unalterable, or predetermined. It does not assert that they are known to anybody, or predictable by scientific means. But it asserts that the future is as little changeable as is the past. Everybody knows what we mean when we say that the past cannot be changed. It is in precisely the same sense that the future cannot be changed, according to metaphysical determinism." (Popper 1982, 7-8)

However, the idea of ontological determinism and epistemic determinism are related. Since the claim of ontological determinism is not empirically testable, it is a tenable theory for any state of the physical world. Precisely for this reason, it is considered too general and weak a theory, and it is difficult to argue for it directly. Since the idea of epistemic, scientific determinism presupposes the idea of ontological determinism, the strongest arguments for the latter are precisely those for the latter.

„Metaphysical determinism is clearly not testable. For even if the world constantly surprised us, and showed no sign of any predetermination or even of any regularity, the future might still be predetermined, and even foreknown by those able to read the book of destiny... Now lack of

testability, or of empirical content, is indicative of logical weakness (not meaninglessness, of course): a doctrine may be logically too weak to be tested. And for the same reason, it may be entailed by some other doctrine. Thus metaphysical determinism is, because of its weakness, entailed by both religious and 'scientific' determinism; and it may be described as containing only what is common to the various deterministic theories. It is irrefutable just because of its weakness. But this does not mean that arguments in its favor or against it are impossible. The strongest arguments in its favor are those which support 'scientific' determinism. If they collapse, little is left to support metaphysical determinism." (Popper 1982, 8)

In view of the above, epistemic determinism and the related idea of deterministic predictability can be defined as follows: a world is deterministically predictable if and only if the world is ontologically determined, and

a. if the cognizing subject knows all the occurring properties of the world at time t_1, then at all time t_2 (in case of $t_1 < t_2$) it is possible for the cognizing subject to know all the occurring properties of the world at time t_2 ;

b. if the cognizing subject knows all the occurring properties of the world at time t_1, then the cognizing subject is able to answer a question about any future property of the world.

In the light of the above, the following observations can be made in relation to Grünbaum's comments. It is of course true that what he calls ontological predictability should not be confused with epistemic predictability, and it is also clear that we cannot draw any conclusions about the former on the basis of the latter. But this also implies that logical predictability must be argued for, and the arguments must be independent arguments, as MacKay shows. MacKay does not attempt to do this at all, nor does Grünbaum. To claim that the states of the system are logically predictable independently of the system is to make an open claim of determinism, and this, without any argument, merely to claim that it is not logically impossible, is not saying much.

The further problem with Grünbaum's comment is that if MacKay's arguments about the cognitive physical system are extended to the physical universe as a whole, and the cognitive system is taken to be Laplace's demon, then one might ask whether it is even possible to imagine a physical cognizer that could ever be capable of more than Laplace's demon. For if it is not possible, then Grünbaum's problem, or indeed any problem of logical predictability, is meaningless, or at least unanswerable.

6.6. Lloyd's Turing test for free will

Specifically in terms of arguing for free will, the argument of Lloyd (Lloyd 2012) can be reconstructed as follows. Assume that the universe is completely determined. Suppose that there are decision problems and that decision systems recursively compute the possible future outcomes of their decisions. Lloyd's hypothesis is as follows: If, in making this prediction, the physical system making the decision is able to simulate its own behavior, then the outcome of the decision will be essentially *unpredictable* to it, completely independently of whether the operation of the universe is otherwise determined or not. The **sense** of freedom of the will, according to Lloyd, is manifested in the fact that physical systems cannot predict the outcomes of their own decisions until they themselves have made the decisions. The sense of free will thus has a dual source. On the one hand, it is caused by the fundamental lack of prediction of decisions, but on the other hand, it is also caused by the resulting surprise of decisions that occur despite the lack of prediction. For this state to arise, the following conditions must be fulfilled:

1. It requires a decision execution system. *A decision—executing system* is any physical system that processes information from input data, using exactly enough data to execute a decision, where the output state is produced according to the decision. A *decision making system* can be a computer or even a thermostat.

   "A decider is anything that, such as a thermostat, takes in the inputs needed to make a decision, processes the information needed to come up with the decision, and issues the decision." (Lloyd 2012, 3607)

   "The first criterion that needs to be satisfied is that the device is actually a decider. That is, the inputs needed to make the decision are supplied to the device, the information processing required for the decision takes place within the device, and the results of the decision issue from the device. Perhaps the simplest man-made decider is the humble thermostat, which receives as input the ambient temperature, checks to see if that temperature has fallen below the thermostat's setting, and, if it has, issues a decision to turn on the furnace." (Lloyd 2012, 3605)

2. It requires a *decision-making process*, i.e. a *decision*. A *decision* or a *decision process is* both a physical process and a recursive *computational process*. These two conditions imply that any decision process is a physical process that can be simulated on a Turing machine. Lloyd also assumes that physical processes can be modeled on a computer or Turing machine.

   "The first assumption that we make is that our deciders are physical systems whose decision-making process is governed by the laws of physics. The second assumption is that the laws of physics can be expressed and simulated using Turing machines or digital computers (potentially augmented with a probabilistic 'guessing' module in order to capture purely stochastic events). The known laws of physics have this feature. These two assumptions imply that the decision-making process can be simulated in principle on a Turing machine." (Lloyd 2012, 3601)

   "Recursive reasoning is reasoning that can be simulated using a Turing machine, quantum or classical. If that reasoning is performed by a system that obeys the known laws of physics, which can be simulated by a Turing machine, then it is encompassed by recursive reasoning."
   (Lloyd 2012, 3604)

   The result of the decision process is in fact the output of a computational process, which in the cases under consideration can be either *'Yes'* or *'No'*, i.e. *'1'* or *'0'*.

   "In our exposition, we focus on Turing machines that simulate a decider's decision-making process... In addition, for simplicity, we restrict our attention to decision problems whose answer is either 'Yes' or 'No'."
   (Lloyd 2012, 3601)

   It is important to note that if every decision process is a physical computational process that can be simulated on a Turing machine, this does not in itself imply that every Turing machine can be implemented by a physical process. Since the occurrence of a *halting problem* (Turing 1936) (Davis 1958, 69-70), on which Lloyd is based, is conditional on the existence of a problem class in which *all* Turing machines are involved, this precondition of Lloyd's is not in itself sufficient for the existence of a *halting problem.*

3. It is also a necessary condition that the subject making the decision is able to simulate its own decision-making process. Lloyd calls this full recursivity. The condition ensures, as Lloyd intends, that the decision system is a *universal* Turing machine realizing system.

4. If the above three conditions are met, the system is unable to predict its decisions: *"Any Turing simulatable decision-making process leads us to intrinsically unpredictable decisions..."* (Lloyd 2012, 3602)

Lloyd's procedure, the *Turing test for free will*, is designed to determine whether a state of a physical system is unpredictable. If the first three conditions are met, the system has reason to feel free: it is in a state where it cannot even in principle know the outcome of its decisions.

Lloyd's philosophical argument is that the unpredictability or unpredictability of physical computer systems as a result of the *halting problem* provides a completely mechanistic explanation of the mechanism of free will, while arguing that whether the physical universe is determinate or not, the absolute unpredictability of our own choices is a consequence of that mechanism — even if the physical universe is determinate. The phenomenal sense of free will is a state of affairs for the cognizing physical subject that arises from a dual source. On the one hand, it is caused by the **lack of** precise **prediction**, but on the other hand, it is also caused by the **surprise of the** decisions thus made. We can examine the time interval between the initiation of a possible prediction decision process and the actual occurrence of the decision. By the above arguments, this time interval *T is* in any case greater than 0. The state in which the predicting physical system is then is precisely the state of ignorance about the decision. Moreover, this ignorance guarantees that, when the decision occurs, the decision will in any case be surprising to the cognizer.

In any case, it should be noted that the source of the subjective experience of free will can be derived from two ideas. One is the idea that our decisions about the future can be changed (E. Szabó 2004, 177-178). If the physical system cannot predict its future decisions, it can think without contradiction that its future decisions are changeable. **This does not, however, give rise to the alternative idea that the source of the experience of free will is the idea that we could have willed our decisions about the present differently in the past.** This does not follow from the problems of unpredictability of future decisions of physical systems (E. Szabó 2004, 177.).

7 Computationally modeling qualia

According to Frank Jackson's knowledge argument (Jackson 2008), Mary, who lives in a black and white world, has all the physical knowledge about the world, yet she has new information when she sees a red apple. If we accept this, then physicalism ─ according to which the description of the world can be realized entirely with the help of physical theories ─ is false. However, even if Jackson's argument about new information is true, we do not have to discard physicalism. Even exclusively physical computer systems are not capable of predicting their own sensory information if they are built using a specific structure and using complexity. Suppose we interpret predictability in such a way that for an agent being unable to predict a specific fact of the world means that the information regarding that specific fact can't be known beforehand, but it can only be obtained either when the fact actually happens or at some later point of time. Then, Jackson's original argument can be understood as a problem regarding the predictive capacities of agents (Sóstai 2024). Based on the core argument of unpredictability, it can be shown even in the case of exclusively physical computer systems that these systems are not capable of predicting some of their own sensory information. First, a question related to sensation information is interpreted as a decision problem with the help of a formal model. After that, in the substantive part, it will be shown what kind of computational system structure and operating conditions are necessary for the creation of decision-related unpredictability. If the fulfillment of the operating conditions of such a system is required for the creation of qualia, then it can not be excluded that qualia can also be realized by a physical computer. It can be shown that a specific type of a computational system can lead to the creation of decision-related unpredictability of a specific type of sensory information. This investigation takes a charitable position towards the concept of qualia to provide it with a physicalist and computationalist explanation.

7.1. Problem description

In his article *Epiphenomenal Qualia* (Jackson 1982, 2007), Frank Jackson describes a thought experiment aimed at showing that qualia are fundamental qualitative sensations that even a person with perfect physical knowledge, i.e. with precise knowledge of the initial conditions and the relevant physical laws, cannot know. If we accept the conclusion of this intuitive description of the problem as true then we must conclude that physicalism, according to which the description of the world can be fully realized by means of physical theories, is false. This intuitive knowledge argument can, however, be explained by a physicalist theory of computer systems, according to which, even in the case of computer systems bounded by the physical Church─Turing thesis (Deutsch 1985) ─ i.e. exclusively physical ─ it can be shown that these systems, given a certain structure and a certain complexity, are not capable of computing their own perceptual information correctly in advance ─ even if they possess information about all the facts of the world. This means that a fundamental unpredictability affects the knowledge of all physical computational systems. I base my arguments in part on Popper's (Popper 1950, 1992) and Lloyd's (Lloyd 2012) arguments about the unpredictability of free will, taking into account their critical revision (Sóstai 2023). For example, unpredictability may occur in the following actual case:

Input > RGB Camera > *Filter* X > Image > Agent

In the example, one element of the computer system (the image-altering *Filter* X) behaves as a black box (Ashby 1957, 86-93) for the agent (Agent M), *which* can also be considered as another element of the system, i.e. its program cannot be directly known, except by examining its output and input states. However, the computation of *Filter* X in the system occurs anyway, only the program of *Filter* X

is not known to Agent M. If in such a case the program of *Filter* X *is* sufficiently complicated, Agent M will be unable to correctly predict the operation of *Filter* X or the associated decision problem.

A question about color information can always be trivially matched with a question about a decision problem. This can be done by asking multiple yes/no questions, e.g. *'What color will image* $U(q)$ *be?'* can be mapped to the decision problem *'Will image* $U(q)$ *be yellow (or any other color)?'*. The decision problem thus poses the question of whether the computational process that the agent perceives to be taking place in any case, where the program of an element of the process is a black box for the agent, i.e. not known, can be predicted in advance by the agent.

The negative response to this can have several causes, revealing deeper and deeper layers of the problem. Firstly, it is due to the problem of induction (Henderson 2022) (Popper 2005, 3-7), whereby the program of *Filter* X cannot be unambiguously determined by examining a finite number of input-output pairs. Even though it cannot be unambiguously determined, it is still possible for *an agent* ─ if it has such a capability ─ to predict it correctly using appropriate heuristics. If, however, *Filter* X contains hidden complexity, hidden parameters (Ashby 1957, 141) ─ especially if the hidden parameters are rapidly and continuously changing ─ or if *Filter* X's program is written in a Turing-complete language (Jones 1997, 227) and the program exploits its complexity, then Agent will be unable to predict *Filter* X's operation correctly.

Because of the undecidability of the Turing equivalence problem (Sipser 2006, 220), it is generally undecidable for Agent M *whether the* program of the filter it predicts is identical to the program of *Filter* X. Since the Turing equivalence problem is reduced to the halting problem, the consequence of the latter*'s undecidability is that Filter* X's program is generally not correctly predicted (Lloyd 2012, 3602). In such a case, the general decidability problem for agent M can be formulated as *'Is it decidable for me whether the sensory information encountered in* $U(q)$ *is the same as the sensory information produced by some program* n *of my own?'*

If the whole system and one of its programs can be of any complexity, we are faced with the general case of the halting problem. This however is not yet enough to consider any such described system as physical Realistically, if we consider only finite, time-constrained computers, where t involves a empirical measurement constraint on the halting behavior of any program n, it can be said that these t time-constrained systems will not be able to correctly predict their own decisions about their sensory information in t time (Lloyd 2012, 3603). This shows that for finite physical systems the problem is undecidable.

Thus, a deterministic, physical explanation can be given for the creation of qualia, that makes an event (e.g., a decision about color information, sensory information) fundamentally unpredictable from the agent's own subjective point of view, even though it is objectively determined. However, since this event does definitely occur, and the agent is able to know this when it occurs, this means that the agent is able to know information that is in principle impossible to predict correctly in advance, given only an accurate knowledge of physical laws and initial conditions. Despite the unpredictability, the predicted states of sensation do occur, and in this case the occurrence ─ at the moment when it occurs and the agent knows what it is ─ has a surprising power. Then we can say that the immediate experience of the state, when it occurs to the agent, will certainly be surprising, beyond the agent's previous knowledge ─ exactly as it arises in Jackson's argument from knowledge. In which case can a physical computer agent (hereafter agent) have unpredictable sensory information?

1. If the system that can compute the sensory information is physical.

2. If an agent capable of processing sensory information is part of a physical system S that is capable of generating predictions about sensory information using hypotheses and is able to answer decision problems about them, such as *'Will the image produced by the filter acting on camera image $U(q)$ be yellow?'* or *'Will image $U(q)$ be modified by filter n?'*

3. If a given piece of sensory information (e.g. color information) cannot be correctly predicted by the agent, even if the prediction is done using empirical measurement constraints.

7.2. The physical system performing the computations

Since it is difficult to define precisely why and when a system, especially a computational system, is physical, we can define such a system by assuming that the physical Church–Turing thesis is satisfied. The definition of the physical Church–Turing thesis by David Deutsch is as follows (Deutsch 1985): "*Any finitely realizable physical system can be perfectly simulated by a finitely realizable universal computer.*" In its traditional form, the Church–Turing thesis is an intuitive conjecture (Copeland 2020). According to this conjecture, a function is algorithmically computable if and only if it is computable by a Turing machine. The thesis establishes a link between the formal definition of the Turing machine and the informal definition of algorithmic computability, and is itself an informal thesis, impossible to prove formally. The physical version of the thesis claims more than that. It can even be interpreted as a potentially falsifiable scientific theory: in physical reality, there are no computers — hypercomputers (Copeland 2020, 5.3.3) — whose capabilities extend beyond the computational capabilities of Turing machines.

The assumption of the physical Church–Turing thesis ensures that the physical decision process can be simulated on a Turing machine. And if it can be simulated on a Turing machine, then the physical systems under analysis can only perform computations that can be recursively computed by Turing machines. Thus all the limitations of Turing machines will apply to physical systems. In this way, if a certain program, namely the halting program, cannot be computed by Turing machines (Sipser 2006, 173-182), then there cannot be a physical computer system that can compute it. The usefulness of the thesis for the line of thought presented in this paper is that, even if the physical Church–Turing thesis is accepted as a strict metaphysical premise, there may exist sentient information for a computer system that is not predictable.

It must be emphasized that in order to imagine the computational system in question as physical and not abstract, it must apply empirical measurement constraints on halting behavior, so it must be able to set constraints such as a time boundary to measure if a computation halts or not. Then, it has to be analyzed if the same type of unpredictability also holds for such a finitely constrained system.

7.3. Formal description of system S

The model of the total system – called system S – builds on the basic concepts of computer science (Hopcroft et al 2006, 1-36) to define a decision problem and an agent M that can in principle solve the decision problem. In computational theory, a problem is usually defined as the question of deciding whether a given string is a member of some language (Hopcroft et al 2006, 31). The present formal model, in contrast, aims to give the problem a semantics, i.e. to interpret a concrete problem of evaluating color information as a decision problem computable by a Turing machine. A problem interpreted in this way will then be shown to be undecidable under certain conditions. A formal description of the general model is given below, with a textual explanation of the concepts and definitions in the description of the specific problem interpreted in the model:

U:

Problem domain.

UP:

Decision problem with specific conditions, question to be decided. The question is related to problem domain U.

L_n _$io1$:

L_n A Turing machine with input-output pairs, where L_n is the *n-th* Turing machine in a certain enumeration.

$INTP(L_n)$:

$INTP(x)$: For every expression x in the physical tape-alphabet of computer L, $INTP(x)$ yields an element y in the domain U. Specifically, for the uninterpreted input symbols of the alphabet of L, $INTP()$ assigns a set of initial conditions in U. For the uninterpreted output symbols of the alphabet, $INTP()$ assigns a set of results in U. $INTP(L\_io1 \rightarrow L\_io2)$, in other words it follows from the definition of $INTP$ that a computation L is now interpreted as computing a certain problem UP.

L_n_$io2$:

This is the nth L_n Turing machine with input-output pairs, which M interprets as the hypothetical solution of *UP*. In other words, L_n is a given hypothesis for a hypothetical solution to the *UP* problem, generated by interpreting the input and output variables of the Turing machine L_n.

M:

M is an intelligent agent with a model capable of theory generation and theory selection. The problem domain of M *is* U. The problem UP is restricted *to* U and M. M can generate a number of hypotheses Ln and then eliminate any false Ln based on a truth function.

$TRUE(L_n, UP)$:

L_n _$io2$ *IFF* UP_io3 each $(io2)$ and $(io3)$ pair, for a given L_n

Variables:

n: the variable representing the n-th Turing machine in an enumeration of Turing machines

x: a variable representing any string, symbol or sequence of symbols in the L_n string

y: a variable representing any element of the formal language of the domain U

$(io1)$: variable representing a single input-output pair for L_n

$(io2)$: variable representing an interpreted single input-output pair for L_n

$(io3)$: variable representing a single initial condition-outcome pair for UP

Sensory information is the output value of an image input calculated by a computer program. Information processing must always be involved in the digitization of physical information (Chakravorty 2018). The operation of a CMOS sensor in a web camera cannot be achieved without signal processing (Mahowald and Mead 1989), (Fossum and Hondongwa 2014). However, these signal processing programs are typically simple, in that their operation is completely known by knowing the program code, and the program code cannot be changed by the computer system (Barkalov, Titarenko, Malgorzata 2019). There is, however, a class of these programs that can be programmed in a Turing-complete language (Jones 1997, 227). Photoshop filters, for example, are typically such programs (Knoll et al. 2003). If the programming language of these filters is Turing-complete, and indeed any algorithm can be written for the filters, then the filters correspond to a Turing machine.

Every computational task related to a transformation (e.g., applying a filter to an image) implicitly corresponds to a computational task representing a decision problem, even if this decision problem is not explicitly expressed. The transformation computational task in this case may be a color–value transformation by the filter (e.g. yellow filter). The computer calculates what a given image will look like after the filter has been applied. The output is a color–related value (*'image q will be yellow'*). However, in addition to the transformation task, there is also an implicit decision task: predicting whether the result of applying the filter satisfies a certain condition or criterion (e.g. *'Will image q be yellow?'*). This secondary task is a decision problem where the output is no longer a color, but a *'yes'* (1) or *'no'* (0). In the example, if the filter is consistently yellow, the implicit secondary computation is always *'yes'* (the image will be yellow), which is a *trivial* decision problem. In more complex cases, where the outcome of the primary task is not simply determined, the implicit secondary prediction task becomes meaningful and complex.

As a concretization of the general model, the specific decision problem $\mathrm{UP1}$ can then be formulated:
'is the filter applied to image q the same as the filter X already applied to camera image U?'
Thus $U(q)$ is the universe of camera images q that already have a certain filter X applied. L_n contains the program of a given filter, which is the nth filter in an arbitrary enumeration. And M is an agent capable of generating and selecting theories. The decision problem thus poses the question whether the computational process X that would otherwise take place in any case, where the program of some element X of the process is unknown, is predictable for agent M. Formally, the decision problem $UP1$ can be expressed in the simplest way, with a single input, as follows:

$UP1(q)$: *'Will the image $U(q)$ be yellow (or any other color)?'*

The instances (unique inputs) of the problem and the inputs of the program L_n to solve the problem are unique q images. A specific L_n is thus a hypothetical solution to the problem, which computes in advance whether applying a filter to q images will result in an image of the specified color. For each image, L_n evaluates $UP1(q)$ to correctly predict the color information decision. This will be a

hypothetical answer to the question posed by $UP1$. Both the problem and the output of the Turing machine that computes the output of $UP1$ are logical binary values: 0 or 1, which can be false or true, but can also be undefined.

7.4. Key concepts

Sensory information, color information: output data of the camera image $U(q)$ that appears as input to agent M after further computation. In this example, it is equivalent to the information content of an RGB image. If the program of L_n is generic and the complexity of the filter program is increased, the sensing information can be not only color information but also any other sensing information. In this case, the *'sense'* may correspond to the input of any peripheral of M, or even to any input of M that is computed as the output of programs for M. Given the appropriate complexity of L_n , they may co–occur and the sensory information may interact.

Detection process: the process of calculating the output value of $U(q)$ from a set of q images. In the case of the system, this can be done directly, or it can be done by prediction, or precomputation, generated by the agent.

Decision value related to sensory information, color information: an output value of $UP1(q)$. The output of $UP1(q)$ is a logical value based on the color result of applying the filter to image q. Thus defined, the output of $UP1(q)$ is no longer just color information, but is now a yes/no answer to a question about color information.

Prediction, forecast, decision: when M generates the hypothesis L_n and then computes the interpreted output of L_n with at least one input q, then L_n forms a forecast. Since L_n represents a decision or answer related to a decision problem, L_n also computes a decision related to $UP1$ at the same time. The output of the prediction or decision is binary, representing a yes or no answer to the decision problem.

Prediction of a decision on a piece of sensory information: a decision problem. Can the value of $UP1(q)$ be computed in advance by a correct program?

Evaluation: Evaluation of $TRUE(L_n\ ,\ UP)$ for any input.

Definition of a system S: Based on the above, S is a system capable of computing sensory information in two ways. S is primarily a system that computes sensory information from a source U(q) using a program X, where the program X is unknown to agent M (black box), but the result of the computation of X is presented to agent M as input sensory information. This sensory information and the decision problems associated with the sensory information in S are computed by X in any case. S is also, in a secondary way, a system for predicting UP decision problems related to sensory information by means of L_n programs, in which Agent M can generate and evaluate L_n programs that hypothetically compute these UP problems.

7.5. When is a decision on sensory information not predictable?

Type S systems exist. In practice, an image processing system interprets the incoming information, the display of any image presupposes a certain amount of information processing (Chakravorty 2018). Therefore, the display of information requires the execution of some computational process.

Self-learning, artificially intelligent agents are also capable of modifying, creating and running programs, generating and evaluating hypotheses (Mohri, Rostamizadeh, Talwalkar 2012). One such system is ChatGPT (OpenAI 2023), in which the problem of unpredictability can be demonstrated experimentally. If, for example, the AI agent is able to detect that its camera image, which previously behaved in a constant, predictable way, does not display the world in the same way as before under new conditions, but has no direct information about why this happens, the agent can infer that there is some filter between the world and the camera image in the computational process. This filter then acts as a black box for the agent (Ashby 1957, 86-93), i.e. it can only deduce what exactly the filter's program is by examining the input-output information pairs.

It follows from the description of the system S that the agent M creates predictions in order to predict the program of its own existing filter. These predictions are based on finite pairs of input-output information, so they are in fact generated in the same way as any empirical theory (Popper 2005, 37-56). In such a case, the consequence of the problem of induction (Popper 2005, 3-7) is that the program of the filter cannot be uniquely determined by finite observation.

As an interesting example of such unpredictability, we can imagine all possible states of an image from a camera with a resolution of only 320 x 200 pixels and a color depth of only 1 bit are 2^{64000}. If the number of atoms in the universe is ~10^{82} = ~ 2^{256} , and we want to physically realize and store these possible states as data, it is understandable why Agent M is not able to compute all possible states, i.e. all possible sensory information, in advance and why Agent M has to guess them by some heuristic.

The problem is further complicated if Filter X contains one or more continuously varying hidden parameters. A hidden parameter (Ashby 1957, 141) in this case means that the output of the filter depends on some environmental variable that is not observable to the system in a constant way. If, for example, the filter varies as a function of temperature, and the agent is exposed to continuously fluctuating temperature conditions, it will receive sensory information indistinguishable from a random program. Under these conditions, the principle of the filter cannot be learned (Ashby 1957, 134). As a concrete example, it is also possible to generate a hidden parameter by basing the color added by the filter on the color of a particular pixel of the camera image, or even on the average color of the image.

7.6. The general case of the prediction problem of sensory information

For general unpredictability, additional conditions must be met. If both the primary computational processes that are directly involved and the secondary computational processes that predict them are sufficiently complex, a system can be created whose state of ignorance corresponds perfectly to the state Jackson describes for Mary (Jackson, 1982), in which something is then missing from her knowledge before she leaves the black and white room.

Filters (Gonzales 2018) are usually only a narrow class of all executable programs. However, in order to be undecidable, it is necessary that the class of programs under consideration is equal to the class of all possible programs, i.e., it is a necessary precondition that the program used to process the

camera signal must be written in a universal, Turing-complete (Jones 1997, 227) program language. The digital filters and image processing plug-ins used in computers and cameras are exactly such (Knoll, et al. 2003), typically software written in C++.

A generally undecidable program might be implemented in the following pseudocode program. For the sake of example, suppose that there exists some standard enumeration of filter programs L_n and suppose that there exists a standard enumeration of program inputs i_n.

Filter Program 2 (def):

Get image 'q'.
n = Rnd(n)
Get 'i_n '.
Get 'Ln '.
If Ln (q+i_n) halts, then color 'q' green.
If Ln (q+i_n) doesn't stop, then color 'q' red.

In general, the problem of prediction is reduced to the problem of Turing equivalence (Sipser 2006, 220). Briefly, the Turing equivalence problem is as follows: given two programs P and Q, do they give the same result for all possible inputs? In theory, we can easily construct a Turing machine Q that always stops. Then suppose that we have a program PEQ *that* checks the equivalence of Q with another arbitrary program P such that $PEQ(P, Q)$ is true if and only if P stops for all possible inputs. Thus, if we had such a program $PEQ(P, Q)$, we could solve the halting problem, which is undecidable. Therefore, $PEQ(P, Q)$ is also undecidable.

It must be emphasized that the problem of Turing equivalence is not a *physical* problem, because in physical reality, no two programs can be thought to be *equivalent*. Therefore in real-life application, this problem is better understood as a *correspondence problem*, especially in the upcoming case, where Agent *M* must utilize empirical measurement constraints on the halting condition of any program. In such a case, two physical programs can be considered *corresponding to each other* if they match each other according to predefined empirical criteria, e.g. they have the same program size or run for the same amount of time.

How is the general case of unpredictability using the halting problem invoked? Interpreting Seth Lloyd's similar conjecture about free will (Lloyd 2012, 3602), and interpreting the given problem in this way as a decision problem, the undecidability of the halting problem is represented in the system as follows. The *n-th* decision unit of the system solving the decision problem corresponds to a program L_n computing a decision on sensory information, which receives the information needed to make the decision based on *q* inputs, and then reaches the decision result *n(q)=(yes)*, *n(q)=(no)*, or no result *(n(q) undefined)*. A prediction is thus in fact nothing more than a decision or a decision process, i.e. a computational process that, for any q inputs of a given program L_n , gives a result *n(q)=(yes)*, *n(q)=(no)*, or *n(q) undefined.*

In the case of prediction using a general program, the variable *n in* fact denotes a class of decisive physical units — all such units, i.e. the entire class of these units — which can be recursively enumerated (Sipser 2006, 170). When can n be any program? When n programs can process not only q *pieces of* image information, but also any other $q'=(q+i)$ pieces of information. When can input q′ be any? This is possible in several cases. For example, it is possible when any other input i can be added to image q, i.e. $q'=(q+i)$. But it is also the case when $q = m(i)$, i.e. input image q can itself be an image generated by a subroutine m from input i. Such a general M agent, performing any n predictions, cannot predict its decision results. Consider the following function f_m computed by the agent's m program: $f_m(n,q) = n(q)$ when L_n stops at input q, and $f_m(n,q)=F(fail)$ when L_n does not stop at input q. Given the undecidability of the halting problem, the result of this $f_m(n,q)$ cannot be computed in general by any m programs of agent M (Lloyd 2012, 3602).

7.7. The finite, physical case of the prediction problem

The preceding argument uses the general halting problem, which is about Turing machines that may run for any number of steps. Physical systems, however, terminate within some finite time. The question is therefore whether the unpredictability result survives the transition from the idealized, unbounded setting to a physically realistic, time-constrained one. We construct the bounded version of the decision problem by letting t define a bound. Consider it as an upper bound on the number of computation steps a machine may execute and also a bound on its program size.

Every decision system in our enumeration is now truncated accordingly: $L_{n\,t}(q) = L_n(q)$ if L_n produces an output on input q within at most t steps, and $L_{n\,t}(q) = 0$ otherwise. Also, failure to answer by step t counts as 'no'. We can now ask whether there exists a uniform procedure that, for an arbitrary time-bounded decision system $L_{n\,t}$ and an arbitrary input q, determines what verdict the system reached — and does so within the same time bound t. Define $f(L_{n\,t}, q) = L_{n\,t}(q)$. This is the function that looks up the answer: given the index of a decision system and an input, it returns whatever that system decides within time t. f is computable: a Turing machine can simulate L_n on q for t steps and read off the result. However, a diagonal argument - mimicking Lloyd's argument in (Lloyd 2012, 3602) - demonstrates that it cannot do that within t:

1. Consider the two-dimensional list A_T whose (L_n, q)-th value is $f(L_{n\,t}, q)$. This list contains all programs $L_{n\,t}(q)$ that can be computed within time t.
2. Let $f(L_{n\,t}, q) = L_{n\,t}(q)$.
3. Let $g(q)=0$ if $f(q,q)=1$, and vice versa. That is, $g(q) = \sim f(q,q)$.
4. If f can be calculated within time t, so can g.
5. But if g can be computed in time t, then $g(g)$ is necessarily equal to $f(g,g)$ (2).
6. And this is a contradiction, since $g(g)$ is defined to be equal to $\sim f(g,g)$ (3).

Since the assumption that f is computable within time t leads to a contradiction, we must reject it: neither f nor g can be computed in time t. A predictor can eventually work out what every t-bounded decision system decided, but only by expending computational resources that exceed the original time bound. This result bears directly on the prediction question that motivates the entire section. The system's original question was: "will the decision system reach a verdict within time t, and if so, what verdict?" The diagonalization argument shows that no uniform procedure can answer this question within time t itself. Any general method — regardless of how it is designed — must exceed the time bound t in order to determine what t-bounded decision systems decide. Prediction, when directed at

an entire class of decision makers operating under a shared resource constraint, is provably more expensive than the decisions it seeks to anticipate. f is a function *about* the set that is nevertheless *outside* the set: f cannot be evaluated within the time bound t, and if we expect f to be total decidable, it cannot exist within the set of all functions A_T, which by construction contains only those functions computable within t steps. Self-reference is therefore blocked — not by some artificial restriction or by fiat, but by the time-complexity gap itself. The function $f_t(L_{n\ t},\ q)$ cannot answer questions about its own behaviour, nor can it serve as a representative of the broader class of t-bounded computations to which it nominally belongs. The conclusion is that t-finite physical computational systems cannot predict the totality of output values of t-finite physical computational systems. The unpredictability that appeared in the general, unbounded case is not an artefact of idealization. It persists under exactly the kind of finite, empirically constrained conditions that characterize real physical computation.

Since only finite systems with measurement constraints on halting behavior can be considered as real physical systems, it is reasonable to ask why it was important to argue for the general case. The reason is that Frank Jackon's set up in the original thought experiment asked what Mary could or could not know in case she had perfect knowledge of all physical facts and all physical theories. If only the finite case would have been argued for, then any non-physicalist defender of qualia could appeal that the finite case of the prediction problem with measurement constraints does not operate with the perfect knowledge of *all* physical facts.

7.8. Demonstrating unpredictability with a thought experiment

To illustrate the above, consider the following thought experiment. Consider a computing physical agent described above, let us call it Mary-demon after Mary (Jackson 2008). Let us consider Mary-demon's knowledge and abilities to be as powerful as those of Laplace's demon (Laplace 2009), with the important remark that Mary-demon is also a physical entity, i.e. it is *itself part of the physical universe* (Popper 2007, 29). Suppose that Mary-demon not only knows all physical facts, but is also capable of prediction along all *possible* scientific theories, i.e. capable of simulating all possible decision processes along any possible physical facts. Consider all possible brain-state configurations of Mary-demon, including all the facts of the world, as simulatable on a Turing machine. Even this system would then be incapable of predicting all future states, given the undecidability of the halting problem (Turing 1936).

Mary-demon's knowledge only covers the decisions that can ever be physically realized. It is important to note that, under these conditions, she does not necessarily realize every single Turing machine, but only some narrow class of them. Since the undecidability of the halting problem requires that, in the general case, the class of Turing machines in the problem includes all Turing machines, this alone would not necessarily make Mary-demon unable to predict her own sensory information. Then the condition that the demon itself is part of the physical universe becomes important. If this is so, then - considering that the physical universe has a finite bound - we are not dealing with the general case of the halting problem, but rather with the finite case described above. The argument then describes a finite physical process that can be used to explain the knowledge gap in Jackson's thought experiment above. Despite the indeterminacy caused by the undecidability of the halting problem, the unpredictable computations of the system do in fact occur, in which case they are unexpected and surprising at the moment they occur and the perceiving physical agent learns that they constitute new - but entirely physical - knowledge. If we identify this state with the appearance of a quale, we get a physicalist, mechanistic, non-intuitive explanation of qualia.

The implications of the unpredictability are thus also related to the philosophy of mind's views on quals. According to Dennett (Dennett 1988), the properties of qualia are privacy, infallibility, inexpressibility, intrinsicity. All of these can be interpreted in the above model. Predictable sensory information cannot be unique, for if it can be predicted, it can be predicted more than once, and moreover, predictable information can be predicted for other systems, provided that predictability presupposes complete knowledge of the program code and its outputs. If the program code is known, it can be copied, in which case the uniqueness of the outputs is lost. In contrast, sensory information that is not predictable, but is nevertheless displayed, can be considered unique. Because of its uniqueness, this information is both completely private for M and subjective (Nagel 1974), in that it is computed exactly (but not in advance) only for M. For the reasons given above, sensory information is also inexpressible in the sense that the program required for its occurrence cannot be determined, nor can it be passed on to any other system. If the program of $filter\ X$ is not directly known, then the program of the entire system S containing $filter\ X$ is not exactly known. An external system K would then be unable to predict the state of S (and M) for exactly the same reasons that M is unable to accurately predict the program of $filter\ X$. Because of the induction problem and the Turing equivalence problem, system K cannot compute what M computes at the moment it computes the output of the filter program. M is also infallible about the appearance of sensory information (Dennett 1988). Since M is the last member of an information processing chain, it cannot be mistaken in the sense that when information appears to it, it already has the modifications it has already undergone. Intrinsicity is the consequence of the fact that the unpredictability does not depend on external factors, but is simply a consequence of the system's own structure.

In an epistemological sense, it misses from M's knowledge ─ since it cannot predict an event, hence it does not know the explanation of the occurrence of the event ─, but this does not necessarily mean that it misses from M's knowledge of the physical world in a metaphysical sense ─ since this predictability would occur even if M really knew everything about the functioning of a determinate physical universe, and the universe really did contain only physical matter. To quote Joseph Levine (Levine, 1983), it cannot be metaphysically excluded that pain (which is a sensory information) is equivalent to the occurrence of some physical process (the firing of C-fibers, or the computation of a filter by a computer that will occur anyway), but if it is, it is a brute, inexplicable fact for agent M.

The sensory information presented to M is determined by the system S, but in principle is unpredictable from the state of the agent M. Thus, this unpredictability reinforces the illusion of independence of higher and lower levels (i.e., consciousness experiences unique to M and correlated determinate physical brain states), i.e., dualistic intuition.

8 Thesis overview and conclusions

8.1 Introduction and problem description

The Church–Turing thesis (Copeland 2020)(Section 2.13), which states that any effectively calculable function can be computed by a Turing machine (Turing 1936), has had a profound impact on our understanding of the nature of computation. When extended to the physical world through the physical Church–Turing thesis – or in short, the PCTT – (Deutsch 1985)(Section 2.18), which asserts that every finitely realizable physical system can be simulated by a Turing machine, it raises fundamental questions about the limits of what can be known and predicted about the behavior of physical systems.

Physicalism (Section 3.1) is the philosophical position that everything in the universe, including mental phenomena like consciousness, can be fully explained by physical matter and energy, governed by the laws of physics. Physicalism combined with the PCTT implies that there are no computations which are not physical computations.

Central to this enquiry is **the halting problem** (Section 2.12). We can say that a computer program – called the halting program H – solves the halting problem if it can look at any other program and tell if that will run forever or not (Crossley et. al 1972, 38-39). Suppose that there exists a halting program H that can calculate the following: $H(\text{‘}P\text{’}, i) = P(i)$, when P stops on input i, and $H(\text{‘}P\text{’}, i) = 0$ when P doesn't stop on input i. Turing's proof of the undecidability of the halting problem shows that no such halting program H exists (Turing 1936).

According to a specific argument (Lloyd 2012)(Section 3.2) called **Argument A**, when the halting problem is considered in conjunction with a variant of the physical Church–Turing thesis, it suggests that there may be physical processes whose outcomes cannot be predicted by any computational means. This assumption frames physical systems as computational systems capable of executing decision making processes. With the assumption of physicalism that only physical entities exist, it can be ensured that any physical decision process can be simulated on a Turing machine. Decisions are then defined as the outcomes of computational processes that resolve specific questions about states of affairs in the physical world (Section 3.3). In physical systems viewed through the lens of the physical Church–Turing thesis, decision-making is equated to executing a computational algorithm that processes inputs to arrive at a conclusion. Predicting the outcome of a computational process, especially in physical systems, involves computing that process's future state based on its initial conditions and its program. Predictions are therefore higher-order computational tasks that determine the results of decision-making processes (Section 3.3). If every physical system can be simulated by a Turing machine, and if there are problems that are undecidable for Turing machines, then it seems to follow that there may be physical states or processes that are fundamentally unpredictable. In particular, some predictions – which are higher-order decisions regarding computational decisions – can't be carried out.

The argument in question – **Argument A** – dealing with the unpredictability of *prima facie* deterministic physical systems has its own history looking back more than 70 years from Karl Popper (Popper 1950) to D.M. MacKay (MacKay 1967) to Adolf Grünbaum (Grünbaum 1971). This thesis delves into the profound implications of this argument, focusing on the core problem that arises when a variant of the physical Church–Turing thesis and the halting problem are taken to hold simultaneously.

The key thesis of this current doctoral analysis is to show that the core argument under investigation — Argument A —, which asserts that there exist physical decisions whose outcome can't be predicted, is false.

8.2 Preliminary analysis of potential philosophical solutions

Since the PCTT doesn't logically imply physicalism, it's possible that the proof of the halting problem can be understood as a kind of platonic truth, while at the same time it seems possible to hold the PCTT as true. However, this interpretation — while popular — doesn't go well with the physicalist intent of Lloyd's argument for purely physical predictors (Section 3.11).

It is also not possible to imagine a hierarchy of universal Turing machines, where M_{n+1} would be able to calculate that M_n can't calculate the n-th halting problem. If this were possible, any undecidable problem could be solved as a consequence (Section 3.12). There is also the question of solving the problem by guesswork. This in itself would not be impossible, but after a certain finite number of steps, a computer that makes a specific decision 0 or 1 — and thus produces a guess in any case — cannot guess the outcome of *any* other computer's decision, if it must make it after any *unspecified* number of steps (Section 3.13).

Another potential solution can be argued as follows: If H is uncomputable, then what Turing's proof entails is that the allegedly representing computation H doesn't halt correctly on each instance of the problem in question. If we followed a physicalist definition of meaning (E. Szabó 2017, 8-9), then we could say that H is not meaningful or that H can't represent the halting problem. This would also be the case for D too, and that is because it uses H's computations as a subroutine, meaning that in order to compute $D('I')$, it would have to wait until $H('I', 'I')$ finishes its computation and halts. This computation however never gets finished, so as an intermediate result, we could conclude that if H is non-computable, then D is non-computable and thus non-meaningful and physically unrealizable either. However, if we take physicalism seriously, we would have to assume that Turing's proof itself is physically meaningful in the sense that it is obtainable or computable by physical agents such as humans or other physical agents such as a physical computer M. Only if Turing's proof itself is physically meaningful would we be able to assert the former intermediate result (Section 3.9).

These attempts show that it is not so easy to resolve the contradiction between Argument A and a physicalist-empiricist world view.

8.3 Main results

The key argument supporting the central thesis of this doctoral dissertation, which is presented against Argument A — Argument B —, concludes that Argument A leads to contradictions (Sóstai 2023)(Sections 3.9 - 3.10)*.*

Argument B (Section 3.9) can be summarized as follows. We assume the Church–Turing thesis. We assume physicalism. We also assume — following Turing's proof — that there exists a program H, which is not effectively or meaningfully computable. This leads to a contradiction, because one the one hand, this implies that a physical and therefore — following the PCTT — computational agent M could calculate that a computation $H(\text{`}D\text{'}, \text{'}D\text{'})$ is not effectively or meaningfully computable. On the other hand, by following Turing's proof, either M itself could not meaningfully represent any meta-metatheoretical decision problem or if such computation by M would be obtainable by H, that would eventually contradict that $H(\text{`}D\text{'}, \text{'}D\text{'})$ is not computable.

The extension of Argument B (Section 3.10) allows for the possibility that H (the halting evaluator) can use the computations of M as a subroutine. This introduces a new layer of complexity and leads to a direct contradiction. If H uses M as a subroutine, it introduces a feedback loop where H depends on M to make a determination about whether certain computations halt. This situation leads to a logical contradiction because of the nature of the halting problem and the constraints of Turing's proof. Suppose that Turing's proof is interpreted by M in a way that $H('D','D')$ does not halt. Now, if H uses M as a subroutine, and M computes $M('D','D') = 0$, then based on the construction of D and H and their dependencies on each others computations, this implies that M determines that $D('D')$ does not halt. Given that $M('D','D') = 0$, H would then evaluate that $H('D','D')$ does not halt. If $H('D','D')$ does not halt, it follows (according to the construction of the program D) that $D('D')$ does not halt. This is because D is designed to behave in a way that if $H('D','D')$ does not halt, $D('D')$ itself does not halt. However, based on the definition of the program D, if $D('D')$ does not halt, then by the setup, $H('D','D')$ should output 0. But if $H('D','D')$ outputs 0, then $D('D')$ is supposed to halt according to H, which contradicts the previous step where we concluded that $D('D')$ does not halt according to H. This shows that the process of using M as a subroutine to H leads to a point where the evaluation of $D('D')$ becomes inconsistent, resulting in a logical contradiction.

The upshot of both Argument B, as well as its extension presented against Argument A is that Turing machines are incapable to even calculate that they can't calculate the output value of any arbitrary program. In Lloyd's terminology regarding computational predictions, if the epistemic capacity of a physical decider is only Turing machine limited — meaning that there is no bound on the measurement of halting behavior according to the PCTT —, then no physical machine can decide that it can't predict a decision. Contradicting Lloyd (Lloyd 2012, 3597), we can say that if humans are physical beings then humans can't even know that they are intrinsically unpredictable to themselves. The problem is that while the conclusion of the original argument asserts that any physical agent limited by the PCTT can't calculate the halting problem, the argument itself presupposes a type of knowledge — the proof of the halting problem — that is not physically computable.

8.4 Investigating a precursor to Argument A: László Kalmár's arguments against the plausibility of Church's Thesis

László Kalmár's critique of Church's Thesis (Kalmár 1959)(Section 4) presents a parallel discussion about a similar recognition of agnosticism ─ similar to Argument A ─, questioning the boundaries of effective calculability and challenging the thesis's completeness.

Kalmár's argument (Kalmár 1959, 74) combined with Hypothesis H along Mendelson's critique (Mendelson 1963, 203-204)(Section 4.4) ─ we can call it Argument K ─ can be summarized as follows: We assume Church's Thesis (or in other words, the Church─Turing thesis), according to which the effectively calculable functions are the recursive functions. We assume that the set of correct proofs are recursively enumerable. We also assume ─ following Kleene's proof or following Turing's proof ─ that there exists a function ψ ─ or a function H ─ which is non-recursive, which is therefore not effectively calculable or computable. Then this leads to a contradiction, because this implies that on the one hand we can provably compute that a statement such as $\exists (y)(\phi(p,y) = 0)$ ─ or $\exists (y)(G(p,y)) = 1)$ ─ is not effectively computable and that on the other hand there is no computable proof that such statements are not effectively calculable.

Kalmár argued that because ϕ can be an elementary function, then $\exists (y)(\phi(p,y) = 0)$ expresses a basic truth about physical reality. If this were the case and according to Kleene's proof $\exists (y)(\phi(p,y) = 0)$ would not be Turing-computable and according to Church's thesis it is exactly the Turing-computable functions which are effective, then it would lead to world view where a basic truth about physical reality could not be effectively obtained. According to Kalmár's views, this would contradict physicalism and empiricism, because Kalmár thought that in physical reality all truths should be effectively obtained.

8.5 Additional, intermediate results

Argument K is similar to Argument B, implying that the set of assumptions used ─ Church's Thesis and a proof of a non-computable function ─ are contradictory (Section 4.7). There are two main differences between Argument K and Argument B. One is the application of the implicit assumption in Argument K that the set of proofs are effectively enumerable versus the assumption of physicalism in Argument B. As an intermediate result, it can be argued that from the point of the arguments under investigation, these assumptions are not so different. If we hold on to the assumption of physicalism in Argument B, then all proofs must be finite, physical processes, which lead from a finite set of axioms and derivation rules through a sequence of formula-transformations to the final formula which is the result of the proof (Crossley et. al. 1990, 14). In a physicalist view of the world, any real, correct proof can and must be a physical process, therefore any real physical proof must be effectively enumerable by a physical computation. This implies that if Turing's proof is considered as 'physically correct' and following the PCTT, we also hold that the set of physically correct proofs is recursively enumerable, then this proof would be included in the enumeration of physically correct proofs. This however eventually leads to a contradiction. This shows that it is the assumption of the enumerability of correct proofs by computations which causes the contradiction between the PCTT and the halting problem.

Another important difference between Argument K and Argument B is that Argument K is used by Kalmár explicitly to argue against Church's Thesis as the definitive source of the contradiction. But it

is quite far from being evident that it is a variant of Church's Thesis, which is the cause of the problem.

The investigation of Kalmár's argument suggests that there are three assumptions in each argument, which are contradictory and therefore each of them can be responsible for the contradiction: 1. a variant of the Church–Turing thesis; 2. a variant of a proof of an uncomputable function; 3. a variant of the assumption that the set of correct proofs are enumerable. If the question is whether it is possible to hold a physicalist worldview without contradictions and we assume that in a physicalist setting, all physically correct proofs are enumerable by some physical-computational agent, then this implies that there must be something wrong with either the variant of the PCTT or the correctness of a proof of an uncomputable function, or both.

8.6 Further problem: the determinability of physical meaningfulness of computations

We have defined two critical conditions (A and B) based on the physico-formalist description of meaning (E. Szabó 2017, 8-9)(Sections 5.1 - 5.4) that establishes a systematic physical and empirical basis for computations to meaningfully represent or predict physical states of affairs. Condition A requires a one-to-one correspondence between computational processes and physical realities, while Condition B demands that these computations accurately reflect the occurrence or non-occurrence of these realities through verifiable outcomes. Meaningful computations must halt with correct outputs and in order for a computation to be meaningful its halting behavior must be empirically verifiable – otherwise it would not be possible to decide if the computation in question corresponds to reality or not (Sections 5.5 - 5.10). This implicit condition relies upon the practical verification of computations against the physical reality. It ensures that predictions are not only theoretically sound but also empirically grounded.

It is shown with explicit examples not only that the physical processes D and H are not meaningful from a common sense physicalist view but also that evaluating halting behavior is crucial for evaluating meaningfulness. Neither D, nor H cannot meaningfully represent the halting problem because they either lead to paradoxes (as shown in the setup with D contradicting H's output) or fail to align with physicalist constraints. D, as an anti-diagonal program, and H, as its evaluator, are shown to be engaged in tasks that inherently produce contradictions under physical law and logical consistency. Also, from a physical realist perspective, every computational process is determined to either halt or not, with no ambiguity allowed. Computations that do not definitively halt are, by this view, inherently meaningless because they cannot produce conclusive, verifiable outcomes. All empirical and naturalist considerations point to the answer that $D(\text{`}D\text{'})$ is not a meaningful computation and that no meaningful physical computation H can exist either.

However, for a computation to be considered meaningful, it must not only halt, but its halting behavior must be empirically verifiable. This requirement aligns with the principles of empiricism, which emphasize observable and verifiable phenomena as the basis for knowledge. By accepting physicalism, it follows that any such evaluation of meaning must be carried out by a physical process.

Suppose we argue that an evaluation of halting behavior is needed to evaluate the halting behavior and therefore also the meaningfulness of a computation (Sections 5.11). Suppose we also accept that the evaluation of halting behavior must be carried out by some physical process. Then the PCTT entails that every physical process, especially the evaluation of halting behavior, should be Turing-computable, which would also entail that the evaluation of meaningfulness of H should also be

representable by some computational process. This however is not possible. For suppose that such a computer M could determine that $H(\text{'}D\text{'},\text{'}D\text{'})$ doesn't halt. Then, because D uses H's computation on a given input value as a subroutine, this would also mean that M could determine that $D(\text{'}D\text{'})$ doesn't halt either. And this would contradict Turing's proof. Therefore, no computation even as powerful as M can determine the halting behavior of H. If every physical process must be computable and no computable process can determine the halting behavior of H, it follows that no physical process can effectively determine this either. This presents a contradiction, where physical processes, which are supposed to be capable of all computable actions, cannot evaluate something as fundamental as halting behavior. This is a significant problem, because neither D and H are meaningful from a common sense physicalist view and that evaluating halting behavior would be crucial for evaluating meaningfulness. The inability to determine halting behavior undermines the entire framework for evaluating meaningfulness in physical computational processes. How can this problem be resolved?

8.7 Further result: An empirically specified, finite, correct variant of Argument A

Argument B implies that even Turing's proof that the output value of $H(\text{'}D\text{'},\text{'}D\text{'})$ can't be computed is also a statement which can't be computed by any physical computation. This implies that no physical validator could determine whether $H(\text{'}D\text{'},\text{'}D\text{'})$ is computable or uncomputable. But if that's true and at the same time there is no solution on how to compute or determine $H(\text{'}D\text{'},\text{'}D\text{'})$, any such solution would just deepen the agnosticism, as the question still remains whether $H(\text{'}D\text{'},\text{'}D\text{'})$ is computable or not. However, Argument B in itself doesn't offer a solution. It only shows that there is something wrong with Argument A. This suggests that it might be the naive application of the PCTT, which disregards what is physically meaningful, which disregards any physical constraints on measurement and which applies an unbounded prediction which can be the source of contradiction of Argument A with physicalism and empiricism.

In an **unbounded prediction**, there are no constraints for the original computations $P(i)$ to be predicted by H. It can be possible that $H(P,i)$ runs for a very long, even indefinite amount of time before H actually finishes its computation regarding $P(i)$. In a **bounded prediction**, this is not allowed, as the prediction should definitely halt for example within some given time-frame. If Argument A ─ which utilizes unbounded prediction ─ is true, then it contradicts empiricism, because it implies that there is a fact of the world ─ $D(\text{'}D\text{'})$'s halting behavior ─ which can't be evaluated for correctness by any physical agent. Therefore Argument A expresses a kind of agnosticism towards physical and empirical knowledge.

A second, key thesis of this current doctoral dissertation is that it is possible to formulate an empirically specified, finitely bounded variant of Argument A, which resolves both the contradictions found in Argument A and the determinability of meaningfulness of computations.

The contradiction in Argument A arises from the problem of predictability in deterministic physical systems and the limits imposed by the halting problem. If a computational prediction is unbounded, then it operates with endless resources and doesn't pose any measurable condition for halting and non-halting. A solution to this problem can be given through the notion of bounded predictions, where the halting condition for these predictions is operationally defined (Sections 5.12 - 5.14). Operational definitions specify the exact procedures or actions required to measure a concept (Bridgman, 1949).

For bounded predictions, operational definitions provide clear observable, measurable criteria for what constitutes a computation halting, a computational prediction being correct, or a physical state being accurately represented by a computational process. To operationalize the halting condition, the halting evaluator H needs to be augmented with the capability to count computational steps and monitor internal states. Setting finite bounds for the time spent on verifying the halting behavior of computations are required for H. This means that for a computation to be considered as halting, it must complete within a predefined finite number of computational steps. This introduces an empirically measurable criterion for halting, whether or not a computation reaches a conclusion within these finite bounds.

A *t-finite* computational process is defined as a computation that is bounded by a finite number of steps and states, denoted by t. A computation or a prediction that is about the halting behavior of another computation within the same *t-finite* class is inherently unpredictable. This is due to the constraints imposed by a finite variant of the diagonal argument of Turing's proof, where a computation cannot predict its own halting behavior or that of another computation within the same bounded conditions without running into paradoxes or contradictions. This unpredictability within a *t-finite* class is a very real consequence which can actually be used to explain why a finite physical agent can feel free or why some sensory information is unpredictable for a physical computational agent, which can give a realistic explanation of qualia. This also allows us to validly state a finite version of Argument A, according to which the following holds: Given the assumption of physicalism and given a variant of the physical Church–Turing thesis, a finitely specified variant of Turing's proof shows that there exist finite physical processes within an empirically defined class t of physical computers which can't be evaluated by a finite physical evaluator H_t belonging to at most class t.

However, there is a critical distinction when it comes to predictions made by a *t+1-finite* physical system about a *t-finite* system. A computational process that operates within a slightly larger bound $(t+1)$ – meaning that it has access to more computational steps or states than the *t-finite* processes it is evaluating – can predict the halting behavior of those *t-finite* processes without contradiction. The *t+1-finite* system has enough computational resources to evaluate the *t-finite* systems' outcomes within the constraints of physical laws and empirical verification.

This distinction resolves the core problem of this analysis, which is agnosticism, i.e. the general predictability dilemma by suggesting that while computations are bound by certain limitations (physical laws, finite resources, and empirical verifiability), they are not universally unpredictable. The above results based on correct empirical specifications if Argument A mean that by requiring a measurable empirical criteria of halting, the problem of unpredictable physical states of affairs disappears. This has the consequence that all computations under empirical criteria of halting can be evaluated for correctness and meaningfulness. Predictions and meaning evaluations about the behavior of simpler or more constrained systems are possible from the standpoint of a slightly more complex or less constrained evaluator. This implies that computational predictability and meaningfulness are not absolute concepts but are relative to the computational resources and bounds of the observer or the evaluating system. This conclusion suggests a hierarchy of computational predictability, where systems can make meaningful and accurate predictions about the behavior of less complex systems. This aligns with physicalist principles by grounding computational processes in the physical universe's laws and constraints and adheres to empiricism by requiring that these predictions should be empirically verifiable.

8.8 A specific application of the results

If Argument A can be understood to pertain only to *t-finite* physical agents, then they can be validly used to explain why physical humans ─ and computers ─ can feel free. Using the former results, according to which a computational prediction which is about the halting behavior of another computation within the same *t-finite* class is inherently unpredictable, it is possible to computationally explain the concept of qualia (Section 7). According to Frank Jackson's knowledge argument (Jackson 2008), Mary, who lives in a black and white world, has all the physical knowledge about the world, yet she has new information when she sees a red apple. If we accept this, then physicalism ─ according to which the description of the world can be realized entirely with the help of physical theories ─ is false. However, even if Jackson's argument about the new information is true, we do not have to discard physicalism. If we interpret predictability in such a way that for an agent being unable to predict a specific fact of the world means that the information regarding that specific fact can't be known beforehand, then it can only be obtained either when the fact actually happens or at some later point of time. Then, Jackson's original argument can be understood as a problem regarding the predictive capacities of agents (Sóstai 2024). Based on the core argument of unpredictability, it can be shown even in the case of exclusively physical computer systems that these systems are not capable of predicting some of their own sensory information. First, a question related to sensation information is interpreted as a physical decision problem with the help of a formal model (Sections 7.2 - 7.4). After that, in the substantive part, it is shown what kind of computational system structure and operating conditions are necessary for the creation of decision-related unpredictability (Sections 7.5 - 7.8). If the fulfillment of the operating conditions of such a system is required for the creation of qualia, then it can not be excluded that qualia can also be realized by a physical computer.

References:

Ashby, W. R. (1957). *An introduction to cybernetics* (2nd impression). London: Chapman & Hall.

Bacon, F. (1902). *Novum Organum*. In J. Devey (Ed.), *The philosophical works of Francis Bacon* (Vol. 4, p. 11). New York: P.F. Collier.

Barkalov, A., Titarenko, L., & Mazurkiewicz, M. (2019). *Foundations of embedded systems* (p. 195). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-12981-1

Bickle, J. (2020). Multiple realizability. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Summer 2020 Edition). Retrieved from https://plato.stanford.edu/archives/sum2020/entries/multiple-realizability/

Block, N. (1978). Troubles with functionalism. In C. W. Savage (Ed.), *Perception and cognition: Issues in the foundations of psychology* (pp. 261–325). Minneapolis: University of Minnesota Press.

Boylestad, R. L., & Nashelsky, L. (2019). *Electronic devices and circuit theory* (12th ed.). Hoboken, NJ: Pearson.

Bridgman, P. W. (1927). *The logic of modern physics*. New York: Macmillan.

Bridgman, P. W. (1949). *The nature of physical theory*. New York: Dover Publications.

Chakravorty, P. (2018). What is a signal? [Lecture Notes]. *IEEE Signal Processing Magazine, 35*(5), 175–177. https://doi.org/10.1109/MSP.2018.2832195

Chalmers, D. J. (1995). On implementing a computation. *Minds and Machines, 4*(4), 391–402.

Chalmers, D. J. (1996). Does a rock implement every finite-state automaton? *Synthese, 108*, 310–333.

Chalmers, D. J. (2011). A computational foundation for the study of cognition. *Journal of Cognitive Science, 12*(4), 323–57.

Church, A. (1932). A set of postulates for the foundation of logic. *Annals of Mathematics, 33*, 346–366. https://doi.org/10.2307/1968337

Church, A. (1936). An unsolvable problem of elementary number theory. *The American Journal of Mathematics, 58*, 345–363. https://doi.org/10.2307/2371045

Copeland, B. J. (2020). The Church–Turing thesis. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Summer 2020 Edition). Retrieved from https://plato.stanford.edu/archives/sum2020/entries/Church–Turing/

Crossley, J. N., et al. (1990). *What is mathematical logic?* New York: Dover Publications.

Davis, M. (1958). *Computability & unsolvability*. New York: McGraw-Hill.

Davis, M. D. (2003). *Computability, complexity, and languages: Fundamentals of theoretical computer science* (2nd ed.). San Diego, CA: Academic Press.

Dennett, D. C. (1988). Quining qualia. In A. J. Marcel & E. Bisiach (Eds.), *Consciousness in contemporary science*. Oxford: Oxford University Press.

Deutsch, D. (1985). Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, 400*(1818), 97–117. https://doi.org/10.1098/rspa.1985.0070

E. Szabó, L. (2004). *A nyitott jövő problémája. Véletlen, kauzalitás, és determinizmus a fizikában*. Budapest: Typotex Kiadó. (pp. 177–178).

E. Szabó, L. (2013). Vázlatpontok a fizikai elméletek fizikalista értelmezéséhez. In Z. Zvolenszky et al. (Szerk.), *Nehogy érvgyűlölők legyünk – Tanulmánykötet Máté András 60. születésnapjára* (pp. 122–129). Budapest: L'Harmattan.

E. Szabó, L. (2017). Meaning, truth, and physics. In G. Hofer-Szabó & L. Wroński (Eds.), *Making it formally explicit* (Vol. 6, European Studies in Philosophy of Science). https://doi.org/10.1007/978-3-319-70632-6_7

E. Szabó, L. (2023). Physicalism without the idols of mathematics. *Foundations of Science*. Pre-print version. https://doi.org/10.1007/s10699-022-09837-0

Ferry, D. K., & Porod, W. (2023). Dissipation and irreversibility in computing. *Materials Research Express, 10*(8), 083001. https://doi.org/10.1088/2053-1591/acebba

Fodor, J. A. (1975). *The language of thought*. Cambridge, MA: Harvard University Press.

Fossum, E. R., & Hondongwa, D. B. (2014). A review of the pinned photodiode for CCD and CMOS image sensors. *IEEE Journal of the Electron Devices Society, 2*(3), 33–43. https://doi.org/10.1109/JEDS.2014.2306412

Gonzalez, R. C. (2018). *Digital image processing*. Hoboken, NJ: Pearson. ISBN 978-0-13-335672-4. OCLC 966609831.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik, 38*(1), 173-198. https://doi.org/10.1007/BF01700692

Gödel, K. (1934). On undecidable propositions of formal mathematical systems. Lecture notes by Stephen Kleene and J. B. Rosser (Princeton University, Institute for Advanced Study).

Grünbaum, A. (1971). Free will and laws of human behavior. *American Philosophical Quarterly, 8*(4), 299–317.

Hartmanis, J., & Stearns, R. E. (1965). On the computational complexity of algorithms. *Transactions of the American Mathematical Society, 117*, 285–306. https://doi.org/10.1090/S0002-9947-1965-0170805-7

Henderson, L. (2022). The problem of induction. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Winter 2022 Edition). Retrieved from https://plato.stanford.edu/archives/win2022/entries/induction-problem/

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to automata theory, languages, and computation*. Boston: Prentice Hall.

Jackson, F. (1982). Epiphenomenal qualia. *Philosophical Quarterly, 32*(April), 127–136. https://doi.org/10.2307/2960077

Jackson, F. (2007). *Epifenomenális qualia* (Á. Polgárdi, Trans.). *Különbség*, *IX*(1), 67–82.

Jackson, F. (2008). What Mary didn't know. In A. Gergely, T. Dávid, G. Forrai, & J. Tőzsér (Eds.), *Collection of texts on philosophy of mind*. Budapest: L'Harmattan.

Jones, N. D. (1997). *Computability and complexity: From a programming perspective*. Cambridge, MA: MIT Press.

Jordan, M. I., & Bishop, C. M. (2004). Neural networks. In A. B. Tucker (Ed.), *Computer science handbook, second edition* (Section VII: Intelligent Systems). Boca Raton, FL: Chapman & Hall/CRC Press LLC.

Kalmár, L. (1957a). Az ún. megoldhatatlan matematikai problémákra vonatkozó kutatások alapjául szolgáló Church-féle hipotézisről. *Az MTA Matematikai és Fizikai Osztályának Közleményei, 7*.

Kalmár, L. (1957b). Abstract of 'An argument against the plausibility of Church's thesis.' In Folder 'nm 49' in Kalmár's Nachlass at the Klebelsberg Library at the University of Szeged.

Kalmár, L. (1959). An argument against the plausibility of Church's thesis. In A. Heyting (Ed.), *Constructivity in mathematics: Proceedings of the colloquium held at Amsterdam, 1957* (pp. 72–80). Amsterdam: North-Holland.

Kalmár, L. (1967). Foundations of mathematics – Whither now? In I. Lakatos (Ed.), *Problems in the philosophy of mathematics* (pp. 187–207). Amsterdam: North-Holland Publishing Company.

Kleene, S. C. (1936). General recursive functions of natural numbers. *Mathematische Annalen, 112*(5), 727–742. https://doi.org/10.1007/BF01565439

Knoll, T., et al. (2003). *Adobe Photoshop application programming interface guide* (Version CS ed.). Retrieved November 28, 2019, from https://UserManual.wiki

Laplace, P.-S. (2009). *Essai philosophique sur les probabilités* (5th ed., Cambridge Library Collection – Mathematics). Cambridge: Cambridge University Press.

Levine, J. (1983). Materialism and qualia: The explanatory gap. *Pacific Philosophical Quarterly, 64*, 354–361. https://doi.org/10.1111/j.1468-0114.1983.tb00207.x

Linnebo, Ø. (2024). Platonism in the philosophy of mathematics. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford Encyclopedia of Philosophy* (Summer 2024 ed.). Retrieved from https://plato.stanford.edu/archives/sum2024/entries/platonism-mathematics/

Lloyd, S. (2012). A Turing test for free will. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 370*, 3597–3610. https://doi.org/10.1098/rsta.2011.0239

Lucas, J. R. (1961). Minds, machines and Gödel. *Philosophy, 36*(137), 112–127. https://doi.org/10.1017/S0031819100057983

Mackay, D. M. (1967). *Freedom of action in a mechanistic universe*. Cambridge: Cambridge University Press.

Mano, M. M., & Ciletti, M. D. (2017). *Digital design: With an introduction to the Verilog HDL, VHDL, and SystemVerilog* (6th ed.). Hoboken, NJ: Pearson.

Matuszka, T., Czuczor, F., & Sóstai, Z. (2019). HeroMirror interactive: A gesture controlled augmented reality gaming experience. In *ACM SIGGRAPH 2019 Posters* (SIGGRAPH '19). New York, NY, USA: Association for Computing Machinery, Article 28, 1–2. https://doi.org/10.1145/3306214.3338554

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of machine learning*. Cambridge, MA: The MIT Press.

Mendelson, E. (1963). On some recent criticism of Church's thesis. *Notre Dame Journal of Formal Logic, 4*(3), 201–205. https://doi.org/10.1305/ndjfl/1093957577

Molnár, Z. G. (2016). Kalmár érve a Church-tézis és a kizárt harmadik elvének plauzibilitása ellen. Retrieved from https://math.bme.hu/~mozow/kalmar.pdf

Nagel, T. (1974). What is it like to be a bat? *Philosophical Review, 83*(October), 435–450. https://doi.org/10.2307/2183914

Newell, A. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM, 19*(3), 113–126. https://doi.org/10.1145/360018.360022

Novikov, P. S. (1955). On the algorithmic unsolvability of the word problem in group theory. *Proceedings of the Steklov Institute of Mathematics, 44*, 1–143.

O'Connor, T., & Franklin, C. (2022). Free will. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Summer 2022 Edition). Retrieved from https://plato.stanford.edu/archives/sum2022/entries/free-will/

OpenAI et al. (2023). GPT-4 Technical Report. *arXiv:2303.08774 [cs.CL]*. https://doi.org/10.48550/arXiv.2303.08774

Penrose, R. (1989). *The emperor's new mind: Concerning computers, minds, and the laws of physics*. Oxford: Oxford University Press.

Piccinini, G. (2011). The physical Church–Turing thesis: Modest or bold? *British Journal for the Philosophy of Science, 62*, 733–769. https://doi.org/10.1093/bjps/axr027

Piccinini, G. (2015). *Physical computation: A mechanistic account*. Oxford: Oxford University Press.

Piccinini, G., & Maley, C. J. (2021). Computation in physical systems. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Summer 2021 Edition). Retrieved from https://plato.stanford.edu/archives/sum2021/entries/computation-physicalsystems/

Popper, K. (1950). Indeterminism in classical and quantum physics. *British Journal for the Philosophy of Science, 1*, 117–133, 173–195. https://doi.org/10.1093/bjps/I.3.117

Popper, K. (1959, 2005). *The logic of scientific discovery*. Taylor & Francis e-Library.

Popper, K. (1972). *Objective knowledge: An evolutionary approach*. Oxford: Oxford University Press.

Popper, K., & Eccles, J. C. (1977). *The self and its brain: An argument for interactionism*. Berlin: Springer International.

Popper, K. (1982). *The open universe: An argument for indeterminism*. London: Routledge (Taylor & Francis Group).

Popper, K. (1992). *The open universe: An argument for indeterminism from the postscript to the logic of scientific discovery*. London: Routledge.

Putnam, H. (1960). Minds and machines. In S. Hook (Ed.), *Dimensions of mind: A symposium* (pp. 138–164). New York: Collier; Reprinted in Putnam (1975a, pp. 362–386).

Rayo, A., with contributions from Rochford, D. (2019). *On the brink of paradox: Highlights from the intersection of philosophy and mathematics*. Cambridge, MA: The MIT Press.

Sedra, A. S., & Smith, K. C. (2014). *Microelectronic circuits* (7th ed.). Oxford: Oxford University Press.

Sipser, M. (2006). *Introduction to the theory of computation* (2nd ed.). Boston: PWS Publishing.

Sóstai, Z. (2023). Analysis of the physical Church–Turing thesis and some philosophical implications of the halting problem. In M. Nemes (Ed.), *Impact points IX. Proceedings of the conference of the philosophy department of the National Association of Doctoral Students* (pp. 145–158). Budapest: National Association of Doctoral Students.

Sóstai, Z. (2024). Modelling qualia with physical computers. *Studia Philosophica Wratislaviensia, 19*(2), 2024. Forthcoming.

Stoljar, D. (2024). Physicalism. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Spring 2024 Edition). Retrieved from https://plato.stanford.edu/archives/spr2024/entries/physicalism/

Szabó, M. (2018). Kalmár's argument against the plausibility of Church's thesis. *History and Philosophy of Logic, 39*(2), 140–157. https://doi.org/10.1080/01445340.2017.1416533

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 42*, 230–265; Erratum in *Proceedings of the London Mathematical Society* (1937), 43, 544–546. https://doi.org/10.1112/plms/s2-42.1.230

Van Kampen, N. G. (1991). Determinism and predictability. *Synthese, 89*, 273–281. https://doi.org/10.1007/BF00413520

Wegner, P. (1997). The paradigm shift from algorithms to interaction. *Communications of the ACM, 40*(5), 80–91. https://doi.org/10.1145/253769.253801

Winograd, T. (1990, April). What can we teach about human-computer interaction? In *Proceedings of CHI 90*. https://doi.org/10.1145/97243.97248

Woodward, J., & Ross, L. (2021). Scientific explanation. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Summer 2021 Edition). Retrieved from https://plato.stanford.edu/archives/sum2021/entries/scientific-explanation/