We should only look for the Computational Mind beyond the scope of the Church-Turing Thesis

PETTERS, Dean and JUNG, Achim

We should only look for the Computational Mind beyond the scope of the Church-Turing Thesis

Dean Petters[a*] and Achim Jung[b]

[a]Sheffield Hallam University, UK [b]University of Birmingham, UK

ARTICLE HISTORY
Compiled January 24, 2024

ABSTRACT
We explore the relationship between computational models in Cognitive Science and the Church-Turing Thesis (CTT). We agree that it is a correct interpretation of the CTT to say that all programming languages are able to implement the same set of computable functions and no programming language can compute Turing non-computable functions. However, we argue it is a misinterpretation and it is not correct to say that due to the CTT, *'all programming languages are equivalent'*. We highlight and explore this and other common misinterpretations of the CTT by considering variations to computational processes that demonstrate a TM does not capture all computational properties that are of interest within Cognitive Science. We show that variations that can give rise to greater computational expressivity than that of a TM include: (i) interaction, (ii) concurrency, (iii) continuity (iv) operating in real time, (v) richer forms of input and output, (vi) indeterminancy, and (vii) intentionality and self-reflection. We argue that since these variations in computational processes lead to formalisms being more computationally expressive than TMs, formalisms that support these variations are outside the scope of the CTT. That is, since the CTT only applies to consideration of Turing computablility, when formalisms possess extra computational expressivity than the expressivity of a TM these formalisms cannot be reduced to TMs. Irreducibility in this sense defines the scope of the CTT and shows the CTT is a fundamental yet isolated phenomenon. We show this approach resolves the Chinese Room Argument and has other benefits for Cognitive Science.

1. Introduction

A number of theories, approaches, and arguments within Cognitive Science and Philosophy of Mind have a direct theoretical relationship with the Church-Turing Thesis (CTT). These include Machine State Functionalism (Kim, 2011; Maley & Piccinini, 2013; Putnam, 1967), Triviality Arguments about Computational Implementation (Sprevak, 2019), computational theories of consciousness (Klein, 2019), and Searle's Chinese Room Argument (CRA) (Preston, 2003b; Searle, 1980). More fundamentally, it is argued that the CTT underpins the whole motivation for the computational theory of mind (Clark, 1991, 2014; Preston, 2003a; Pylyshyn, 1979). We argue that the relationship between Cognitive Science and the CTT should be re-assessed due to a need to re-interpret how the CTT is invoked in Cognitive Science. The CTT is commonly interpreted as stating that the intuitive concept of computability is fully captured by the Turing machine (TM) or any equivalent formalism (such as recursive

*Corresponding author. Email: d.petters@shu.ac.uk

functions, the lambda calculus, Post production rules, and many others). The CTT implies that if a function is (intuitively) computable, then it can be computed by a TM. Conversely, if a TM cannot compute a function, it is not computable by any mechanism whatsoever. We accept and don't challenge these aspects of the CTT. That is, we do not propose any form of hypercomputation[1]. Rather, the point we make is the other way around. When a machine carries out sophisticated computational tasks, then standard Turing machines (TMs) and their mode of operation are not sufficient to explore the range of computational possibilities. So it is not correct to say that aside from issues of computational complexity, due to the CTT, *'all programming languages are equivalent'*, and *"no model of computation more expressive than Turing machines can exist"* (Goldin & Wegner, 2005, p. 152), or that TMs provide a *"treatment of the limits of mechanism"* and *"the universal Turing machine can simulate the behaviour of any machine"* (Copeland, 2017, section 2). We suggest that what these misinterpretations have in common is the incorrect view that the kind of computation that the CTT covers is all that is needed for any computational task. We suggest this this incorrect interpretation arises from taking a solely function oriented view of computation. The key point we make is that whilst computational formalisms do not differ in terms of Turing Computability, variations between computational formalisms do differ in expressive power (expressivity/expressiveness) (Felleisen, 1991), in a way that is relevant to Cognitive Science models[2]. The expressive power of a computational formalism is the processes and structures that can be represented and communicated in that formalism (Felleisen, 1991). When computational formalisms possess variations that mean they can be more expressive than TMs, we describe those formalisms as being beyond the scope of the CTT.

When considering the difference between computation within and beyond the scope of the CTT, we emphasise that TMs and other computational formalisms within the scope of CTT are closed systems with a single thread of control that take a fixed finite string as input. Output (if it is produced) will be a finite string. Table 1 presents variations to TM processes that are relevant to Cognitive Science. Our key claim is that computational formalisms that are interactive, concurrent, continuous, with indeterminancy, real-time computation, richer input, and intentional computation and self-reflection, have increased computational expressiveness. This means they can represent and communicate additional objects, entities, processes and relations compared with those represented and communicated by TMs and equivalent computational formalisms (Sloman, 1996, p. 3). These additional objects, entities, processes and relations include networking of interacting persistent agents that are more intimately integrated to real world sensors, with a richer abstraction of time and space, and which are capable of self reflection. We argue these additions can lead to a new class of observable behaviours that are important for explanations of mental phenomena and cannot even be specified let alone observed within the scope of the CTT (Schächter, 1999, p. 38).

Computation within the scope of the CTT (such as a TM) refers to computation of the type that can be carried out as a 'batch process' where input is provided at the start (historically on cards) and output is returned on a printer. Since 'batch computation' refers to a technological aspect of this style of computation it is not an ideal term to use to describe this class of process. We might also use the terms: function-

[1]The aim of hypercomputation (also known as 'super Turing computation') is to find input-output processes that compute functions that cannot be realised with a Turing machine (Copeland, 2002))

[2]Computational formalisms do not differ in regard to Turing computability. They do differ with regard to complexity. However, this is limited as TMs and alternatives that are outside the scope of the CTT compute within polynomial time of each other. Our argument is about expressive power not complexity

based; mathematical view; classic; closed; static; finite input and finite output; serial (in the sense of non-concurrent); and non-interactive (in sense of not parallel and no interruptions). From these alternatives, here we choose to refer to computation within the scope of CTT as 'input-output computation' as this is a neologism and avoids confusions that might occur with some of the other possible labels set out above. For example, we avoid referring to input-output computation as function-based due to possible confusion with functional programming which is not what we are referring to and can in fact be high-level and interactive. For the alternative to 'input-output' computation we choose another neologism. Alternatives are: full; open; general; dynamic; comprehensive; real world; concurrent; interactive; and concrete. Of these we will refer to computations that are not input-output computation as 'general computation' as we are looking for an umbrella term that captures that what we are referring to is all computation that is not input-output computation. So referring to it as just 'interactive' or just 'concurrent' misses out that we are also referring to computation that might continue on in a 'forever' process, has a less abstract representation of time, allows richer kinds of inputs and outputs, involves indeterminancy, and intentionality and self-reflection.

This terminology 'input-output computation' and 'general computation' is not postulating a new as yet unseen or untested approach. Rather it is giving new terms to computational approaches which already exist. Goldin and Wegner (2005), examine the likely origins of the incorrect belief that TMs are a maximally expressive computational formalism, which they term the 'Strong Church-Turing Thesis'. They explore the reasons why this belief holds such sway. They suggest that the way the first generation of computing machines were designed and used (i.e., the "batch processing" discussed above) was so strongly correlated with Turing's mathematical concept of a (human) "computer" (i.e., his "Turing machines") that standard undergraduate textbooks adopted Turing machines as a suitable formal abstraction of computing practice (all computing practice!). In section 2 we show TMs were created to answer questions of effective calculability and formal derivability and other questions that can be solved by what a mathematician can do with a paper and pencil (MacLennan, 2009, page 369). This leads TMs and similar formalisms within the scope of the CTT to exist in a particularly constraining 'frame of relevance' (MacLennan, 2009, page 369). If computational questions being posed include consideration of the effect of variations listed in table 1 then there is a different frame of relevance and general computation formalisms rather than input-output formalisms are required. Wegner (1998) makes a related point that the TM model gains formal tractability but at the cost of becoming too strong an abstraction and *"abstracting away the ability to model autonomous external events, throwing out the baby with the bathwater"* (Wegner, 1998, p. 317). In section 2, we develop these ideas, considering how the historical development of the CTT supports the claims we make about the constrained scope of the CTT. In section 2 we also discuss in more detail how the variations described in table 1 can give rise to greater computational expressivity. Thus demonstrating that computational formalisms with these variations are beyond the scope of the CTT. Section 3 includes a diagrammatic presentation of these ideas. A review of the literature on misapplications and misunderstandings of the CTT follows in section 4. We emphasise, this issue is not just relevant to the application of the CTT to Cognitive Science but also to contemporary computing practice. Situations in contemporary computing are now so rich, they can no longer be said to be covered by a paradigm where the inputs are known in advance, the system is left alone to do its computation, and then provides the answer.

Computation within the scope of the CTT does provide adequate explanations for some cognitive phenomena. For example, where a cognitive phenomenon is adequately captured by a function which is computed from finite input in a closed serial process that abstracts away from autonomous real world events, real world sensors, time, and space, and with finite output. However, this approach is not suitable for capturing very many other important phenomena of interest for the computational mind. Schlesinger and Parisi (2001) present a contrast which is helpful in considering this issue. They distinguish between two ways in which cognitive models can sample from an environment. In offline sampling, each input sensory pattern represents a discrete, independent moment in time. After each sensory input the models can output an action and it can receive a feedback training signal. However, the model does not experience any consequences of that response. Either immediately or in the long-term. Schlesinger and Parisi (2001) note that this state of affairs is analogous to a typical tightly controlled psychology experiment where independent, discrete training exemplars are presented in randomised order and interference between successive training exemplars is treated as noise. In contrast, online sampling (active sampling) samples inputs as a function of the model's output. So each response the model makes has an iterative effect on where the model is located in the input space. The models's own output influences the next input it receives. Where offline sampling abstracts away from real time and does not include moment to moment sequences of sensation and action, online sampling engages far more closely with the real-world. Online sampling captures temporal structure and causal processes in contingent moment-to-moment sequences of action and sensation. New sensations are influenced by previous actions, and then these sensations influence further actions. These causal relations are simply absent when offline sampling is used. This leads to a different way of presenting the main claim in this paper: whilst input-output computation may be able to model data gained from offline sampling, only general computation can represent the relations, causal processes, and contingencies involved with online (active) sampling. It is important to clarify that more expressive computational formalisms beyond the scope of the CTT still compute just the same set of computable functions as those within the CTT. Also, they don't compute any Turing non-computable functions. The additional expressivity is on top of what formalisms within the scope CTT can do.

In this paper we show that the misinterpretation of the CTT that views TMs and other input-output computational formalisms as maximally expressive can lead to a problematic form of reductionism. It is reductionist because this misinterpretation takes an approach to computation that views TMs and equivalent input-output computational formalisms as as powerful as more expressive computational formalisms. So leading to explanations for complex phenomena in Cognitive Science in terms of the simplest basic processes. It is problematic because the computational equivalence is not justified. As it ignores extra computational expressivity that results when computations involve interaction with the environment, being concurrent, existing in time etc. For example, we argue that phenomena such as consciousness and meaning in Cognitive Science need to be explained by more than the operation of TMs or equivalent input-output computational formalisms. The reductionism is apparent in concrete implementational terms: machines for interactive processing, networking, or multi-processing will involve more complex hardware than that for 1950s batch-job processes with paper cards for input and printers for output. At this hardware level the operation of machines that support these kinds of capability cannot be reduced to input-output computational mechanisms.

In sections 5 we show the 'pay-off' for Cognitive Science in adopting a corrected view

of the scope and applicability of the CTT. We consider benefits of our re-appraisal for how the CTT should be interpreted for: the Machine State Functionalism Approach; and for Triviality Arguments about Computational Implementation. We apply the thesis put forward in this paper criticise to invocation of TMs in consciousness research. Then we re-appraise Searle's Chinese Room Argument (CRA) in light of our claims about the scope and applicability of the CTT. We show that the CRA does apply to input-output computation and it does apply to some examples of general computation. However, importantly, we show that the CTT cannot be applied to all examples of general computation. As general computation is beyond the scope of the CTT. Thus defeating the key purpose of the CRA.

1.1. An explanatory analogy that clarifies our argument

So far this introduction has coined several new terms and phrases, including: *'scope of the CTT'*, *'input-output computation'*, and *'general computation'*, as well as using the term *'irreducible'* in the particular context where the greater expressive power of computational formalisms beyond the scope of the CTT means these formalisms are irreducible to TMs. Therefore, this subsection will clarify these terms by presenting an explanatory analogy comparing the structures and properties of computational systems with structures and properties apparent in the architecture of building design.

Architectural analogies in Computer Science and Cognitive Science that compare information processing with building design and use are not new. As Anderson (2009, p. 4-7) recounts, Allen Newell introduced the term "cognitive architecture" into cognitive science in analogy to the term "computer architecture" (Bell & Newell, 1971). Which itself was an analogy to the architecture of buildings introduced by Fred Brooks (1962 cited in (Anderson, 2009)). Anderson (2009) notes that the meaning of "architecture" as used by Brooks and then by Bell and Newell means the product of design. That is, both computer architectures and cognitive architectures are descriptions of structures and processes that can achieve particular information processing functions. In the same manner as an architect of buildings specifies how structures used by builders achieve the functions desired by the dwellers of buildings.

Here we elaborate the analogy between the architecture of buildings and information processing architectures by adding some more detail and constraints. Buildings such as houses are composed of various components - bricks, joists, windows, pipes etc. Computational formalisms have elements such as input-output computation and other elements such as interruptions, concurrency, continuity, real-time computation, richer input, indeterminancy, and self-reflection, that combine together to create more expressive computational formalisms. How should we integrate the CTT into an analogy between buildings and computational formalisms? We can introduce the CTT into the analogy as a constraint about the equivalence of bricks. A 'CTT for bricks' might say that it doesn't matter whether your bricks are made from clay or from concrete or from natural stone or whatever, they all have an equivalence in their physical properties which means they can be used to build houses. All bricks share an equivalence as some critical properties of all bricks are in common. This is in analogy with the equivalence that holds for input-output computational formalisms is they all produce the same set of Turing computable and Turing non-computable functions. The key point is that in this analogy the CTT applies to bricks not houses. We emphasise, for this analogy to fit the argument made in this paper, there is no 'CTT for houses'. There are many different kinds of houses and there is no simple equivalence between houses

Domain of variation to TM processing	Does it change Turing comput-ability?	Description
Interaction	No	ongoing interaction with the outside world: in frequency of interruptions; interleaving and dependencies between interruptions and ongoing processes; and in the nature of the information provided by interruptions
Concurrency	No	the process view of computation; parallel and concurrent computation, and multi-processing; including asynchronous parallel computational processes with independently varying speeds; continuous processes, and non-discrete processes
Continuity	No	the kinds of 'forever' processes that are found in operating systems;
Real time	No	more detail in how the computational system exists in time and the real world, with real-time computation
Richer input	No*	richer forms of input such as real world sensors, real number computation, continuous input, and input with infinite inputs (streams)
Indeterminancy	No	true random numbers
Self-reflection	No	ways in which higher-level and non-reducible routines within a computational system can analyse basic programs for 'meaningful' patterns in their own internal processing (self-reflection) including intentional computation in a higher-order computational setting.

Table 1.: Showing 7 ways in which TM processing can be varied to increase computational expressive power. These variations are all important for Cognitive Science models. Whilst none of the 7 variations to processing changes Turing computability, each of these 7 variations can change the expressive power of computational systems (*we do not claim that richer input leads to super Turing computation. However, others have made this claim (Cabessa & Siegelman, 2014; Siegelman, 1995, 1999)).

in the way there is between bricks. Just as there is no 'CTT for general computation'. As formalisms for general computation are beyond the scope of the CTT.

Elaborating the analogy further we can consider that while understanding bricks is fundamental for becoming an architect, it is however clearly not sufficient. And this is for (at least) two reasons: Bricks can be combined into larger structures of surprising properties, for example, they can be arranged in the shape of an arch and thus can

form 'bridges'. Nothing we know about bricks on their own (their 'CTT') tells us anything about the amazing properties of bridges. Secondly, bricks are combined with other functional elements such as windows, joists, pipes, and many others to create 'houses'. Again, the task of the architect in planning a house can not be reduced to her expertise in bricks. In our analogy when bricks are incorporated in houses they keep their brick properties. Just as formalisms for general computation maintain the capabilities of input-output formalisms - they compute the same set of computable and non-computable functions that input-output formalisms do. Another way of presenting this idea is that the kind of computation that the CTT talks about is only a 'building block' for the computational 'systems' that have been created for many decades now and which contain several other 'building blocks'.

A key lesson from this paper is that we shouldn't give up on explaining phenomena like consciousness or meaning in computational terms because it can be shown a TM is incapable of producing these phenomena. In our analogy we can think that the domain of properties, functions, and basic capacities for bricks is different to that for houses. For example, bricks have compressability, colour, and surface texture. Where for houses we can also consider how their architecture helps fullfil the functions demanded by those who dwell in them. We evaluate houses differently to bricks because houses belong to a different domain of experience and function. In an analogous manner we can consider how properties, functions, and basic capacities may differ for cognitive models created respectively with input-output and general computation. In ways that are relevant to creation of computational

2. Overview of the Church-Turing Thesis

The central argument of this paper is that invoking a mathematical theorem to make inferences about real-time physically instantiated systems should be done with careful consideration of both the scope of the theorem, on the one hand, and the properties and complexity of the physical system, on the other. More specifically, we take the position that there are implementable (and widely implemented) algorithms which are outside the scope of the Church-Turing Thesis (CTT), and that this needs to be taken into account when invoking the CTT in contexts in Cognitive Science such as the CRA. To explain our argument, in this section we begin by presenting the historical background to the CTT, examine its scope, and then consider its misapplications and misunderstandings.

2.1. Historical background

2.1.1. Hilbert's questions and Turing's analysis

While today it is reasonable to define mathematics as an "activity involv[ing] the use of pure reason to discover or prove the properties of abstract objects" (as Wikipedia does, Wikipedia Contributors (2022)), for many millennia the task of the mathematician was to find *procedures* for solving precisely presented problems. Thus we have the "Euclidean algorithm" for finding the greatest common factor in a pair of numbers, the "Chinese remainder theorem" for solving simultaneous congruences, the "Newton-Raphson method" for approximating a root of a differentiable function, and so many others. It was therefore natural for David Hilbert in his famous lecture in 1900 to ask for a "process by which it can be determined by a finite number of operations whether

[a Diophantine] equation is solvable in [...] integers" (his "10th problem"). Nobody at that time could have anticipated that such a "process" does not exist and that it would take 70 years to arrive at this conclusion.

In 1917, it was again Hilbert who asked for a "process" that would allow one to decide whether a given formula is the logical consequence of a given finite set of axioms. Again, this was considered a valid question and it became known as the "Entscheidungsproblem" (decision problem) among mathematicians and logicians.

For the purposes of the present paper it is crucial to emphasize the general form of both Hilbert's 10th problem and the Entscheidungsproblem: In both cases, the formulation of an instance of the problem can be given in a finite way, by writing down the equation, respectively, by writing down the finitely many axioms and the formula under consideration. The "process" is expected to manipulate these representations and yield an answer after finitely many steps. It follows fixed rules and proceeds in an uninterrupted fashion. We may view the formulation of the given instance of the problem as the *input* to the process and its eventual answer as the *output*. To distinguish this mode of "computation" from the others we will consider below, we call it an "input-output process" or "input-output computation".

In his famous paper of 1936, (Turing, 1936), Alan Turing proved that there can be no input-output process for the Entscheidungsproblem. Three ingredients are at the heart of his argument: Kurt Gödel had shown in 1931, (Gödel, 1931), how a finite sequence of steps in a formal system could be encoded by a number ("Gödelization"), thus be made into arguments for the process itself. This was yet another application of the "diagonalization method" which had originally allowed Georg Cantor to conclude that there are uncountably many real numbers. Alan Turing added to these two a careful analysis of what it means for a human "computer" to execute the steps of a precisely specified process. Crucially, he showed convincingly that those steps could be broken down into finite sequences of very elementary operations: reading or writing one of three possible symbols on a linear medium (now called the "tape") and moving attention from one cell of the tape to an adjacent one.

Input-output processes lend themselves to a straightforward mathematical *semantics*: The finite input to the process can be viewed as a natural number and likewise the output. Thus we get a *function* from $\mathbb{N}$ to $\mathbb{N}$, the only proviso being that the process may run forever without producing an output, so more precisely then, the semantics of an input-output process in the sense of Hilbert is a *partial function* from $\mathbb{N}$ to $\mathbb{N}$.

Contrast this natural and precise semantics with the (only slightly more complicated) situation where we use a given "finite process" in the sense of Hilbert, twice in a row. It is now no longer clear how the semantics of this should be expressed mathematically. It is certainly not the case that we are dealing with a (partial) function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N} \times \mathbb{N}$ as the process during its first invocation does not "know" about the argument it will be given in the second invocation, so its answer can not depend on that second argument. Furthermore, at the time of the second invocation it is not clear whether the process is allowed to take into account the outcome from the first invocation, in other words, whether it has internal persistent memory within which to record the inputs and outputs from previous runs. In the settings that Hilbert was concerned with, internal memory is of no importance, yet the fact that modern computers have access to persistent internal memory is essential to our daily interaction with them. This difference between the input-output process viewed in isolation and the reality of input-output processes being repeatedly invoked during interactions with computing machines is at the heart of the present paper.

2.1.2. The Church-Turing Thesis

In his 1936 paper, Turing expended considerable effort in arguing that his analysis of Hilbert's notion of "finite process" is correct and that his "Turing machines" capture faithfully the steps that a human "computer" would go through in executing such a process. Thus he was able to say that a (partial) function f from $\mathbb{N}$ to $\mathbb{N}$ is *computable* if and only if there exists a Turing machine whose semantics (input-output behaviour) equals f. There was and is no way to prove Turing's assertion since it refers to a necessarily informal concept, namely, what a human may be capable of computing. However, there is now overwhelming empirical evidence for it. Other formalisms, some of which were not even intended to capture the intuitive concept of "finite process", have been shown to yield exactly the same class of computable functions:

- 1933 - Kurt Gödel's and Jacques Herbrand's *general recursive functions*
- 1936 - Alonzo Church's λ-*calculus*
- 1943 - Emil Post's *production systems*

On the basis of these observations, Stephen Kleene was willing to declare that Turing's analysis was correct and complete, and hence that the notion of "computable function" is absolute, and can be defined by any of these mechanisms (or indeed any of the hundreds of mechanisms and programming languages studied since). This is the "Church-Turing Thesis" (CTT). Using our terminology from above we can state it as follows:

Church-Turing Thesis (formulation 1): Every intuitively executable input-output process can be expressed as a Turing machine.

With reference to the mathematical functions from $\mathbb{N}$ to $\mathbb{N}$ we can also state it as follows:

Church-Turing Thesis (formulation 2): Every (partial) function from $\mathbb{N}$ to $\mathbb{N}$ that can be effectively computed in some way, can also be computed by a Turing machine.

Or in more modern language:

Church-Turing Thesis (formulation 3): As far as functions from $\mathbb{N}$ to $\mathbb{N}$ are concerned, all programming languages are equally expressive.

2.2. Scope of the CTT

The CTT has been tested in many directions. For example, it is possible to simplify Turing's elementary operations even further so that all that is required are two counters (of unlimited capacity), the ability to increment and decrement the counters by one, and to be able to recognise when a counter contains zero. In the other direction, one might imagine it helpful to have more than one tape, or a two-dimensional tape, more characters to put into individual tape cells or to be able to be "aware" of the contents several cells at any one time. All these augmentations are easily shown to be reducible to the standard model. Carrying on in the same vein, it is then very easy to show that a programming language is *Turing-complete* (capable of expressing all Turing machine operations). All that is needed is access to (potentially) unlimited memory plus minimal control structures. For unlimited memory, pointer structures are

typically considered a valid approach although strictly speaking, they too have finite limits in practice. In order to show the other direction, it suffices to demonstrate that Turing machines are capable of simulating the operation of a standard von Neumann architecture, since programming languages are designed to be compiled to code that runs on such hardware.

Given these considerations, no one really questions the CTT, and *we are no exception.* It is rightly taken as a cornerstone of theoretical computer science and a core element of the computer science curriculum. However, one must not forget that the CTT is concerned only with input-output processes, that is, with situations where the input is given as a finite string of symbols in some formal language and where a single response is sought from the computational process, or in other words, with the effective computation of a (partial) function from $\mathbb{N}$ to $\mathbb{N}$. Thus it is *correct* to say that all programming languages are able to implement the same set of computable functions but it is *not correct* to say that "all programming languages are equivalent" *without* the qualification about numeric functions. That is because the CTT does *not state anything* about computation more generally, and even more strikingly, *there is no analogue of the CTT for general computation.*

The misconception that the CTT applies to all forms of computation is very widespread. Both within Computer Science (MacLennan, 2009; Schächter, 1999), even Theoretical Computer Science (Goldin & Wegner, 2005; Jay, 2019), and also outside the computer science community (Copeland, 2017). We consider the nature of the misconceptions about the CTT held in different fields in section 4. In the next subsection we consider the ways in which computations can be beyond the scope of the CTT and why this is important to Cognitive Science

2.3. Illustrating that general computation has no analogue of the CTT

This section demonstrates benefits for Cognitive Science by correcting the misinterpretation of the CTT that TMs are a maximally expressive computational formalism. We do this through computational scenarios that illustrate there is no analogue of the CTT for general computation.

2.3.1. Interaction

Turing did very briefly mention in his 1936 paper machines that can receive input after they are already running. He termed these 'choice machines' and these are completely peripheral to the concerns of the 1936 paper in solving the Entscheidungsproblem (Petzold, 2008; Turing, 1936). So it is clear from the 1936 paper that what have come to be termed 'Turing Machines' were never intended by Turing to model interactive processes. In illustrating how interaction leads to computations outside the scope of the CTT, we start with the example alluded to in section 2.2, namely, the use of an input-output process more than once. This is probably the simplest extension to the classical situation one can think of and it is definitely not covered by a single call to an input-output process, yet it is ubiquitous in everyday interaction with computing devices. A standard Turing machine that answers each request is the simplest computational model for this situation. However, as we already said, it is clearly *more powerful* to have a device that has persistent memory in which it can record previous interactions. Such a machine can not only avoid recomputing answers to questions it has seen before, but it can also keep a tally of the questions it has seen most often. Is this extension the best we can do? Definitely not, because we could now add a facility for the machine to

be able to erase the record, and thus return to its virgin state of seeing questions anew. Or perhaps it has a "two-stage" memory where it keeps absolutely everything in the second stage while being able to reset the first stage, plus the capability of restoring a previous state from the secondary to the primary memory. Aside from recording previous answers, such a machine could be allowed to *adapt* its answers according to previous interactions. Such *learning* behaviour is no longer covered by Hilbert's idea of an input-output process for a single fixed function. Learning mechanisms are clearly of relevance in myriad contexts in Cognitive Science.

Task switching is another important example of a interactive processing that is outside the scope of CTT and vital to natural computation. For example, Monsell (2003) highlights the delicate balancing act in cognitive systems between between multiple concurrent tasks, and dealing in an appropriate timely manner with potential interruptions in real time processing. Monsell (2003, p. 134) notes, efficacious moment to moment cognitive performance requires mastery of the complex interplay of potential actions. Taking into account the availability, frequency and recency of alternative tasks alongside the significance of the context and any other relevant external and internal stimuli. Thus interaction with the environment requires delicate 'just-enough' calibration that balances the possibility of unnecessary disruption with flexibility to make opportunistic responses to changes in the environment. TMs and other input-output computational systems cannot provide this kind of response. For example, since situations within the scope of the CTT are closed and programs solely act as implementations of mathematical functions, they *"shut out the world while computing"* (Wegner, 1998, p. 316). Thus demonstrating the limitations of using the TM model of computation as a standard computational model within Cognitive Science and the Computational Theory of Mind.

2.3.2. Concurrency

We can extend the idea of continued interaction by considering a whole *network* of computational agents which communicate via "channels", in other words, *distributed computation*. Once again, there is a whole spectrum of possibilities as to what kind of interactions are allowed in such a network. Is all communication between two agents only, or is communication via a shared medium allowed? Do messages stay on the channels forever until they are "consumed" by the intended receiver? Is an agent blocked from further progress if it seeks a response from another agent but this response is not delivered? What happens to the arrival of data that was not requested? Are agents allowed to establish new (or drop existing) connections? And on and on. It has proven extremely difficult to pin down mathematically what a network actually computes, thus it has so far been impossible to compare different approaches to distributed computation as far as their computational power is concerned; there is no CTT for it. Put another way, sequential algorithms cannot expressive concurrent processes (Wegner, 1998). For example, the Calculus for Communicating Systems (CCS) (Milner, 1980) and Communicating Sequential Processes (CSP) (Hoare, 1985) are two styles of describing parallel systems. Both are very powerful calculi about which the CTT has nothing to say. Thus we cannot rely on the CTT to determine if CCS is as expressive as CSP; both were invented for a distributed world and there is no new CTT for this setting. Yet, distributed computation has been a fact of life for decades, and even more, most computers are stymied in their operation if they are not connected to a multitude of servers via the Internet. Returning to the topic of this article, we are well aware that language learning happens within a *community of human beings*, with

multiple interactions going on all the time. Concurrency is clearly vitally important to Cognitive Science explanation of information processing in natural systems such as humans and other animals. For example, contemporary cognitive architecture possess high levels of concurrency (Anderson, 2009; Newell, 1990). The operation of cognitive architectures would make no-sense if shoehorned into a serial processing paradigm.

2.3.3. Continuity

Computational processes that have continuity are of critical relevance to Cognitive Science. An example of continuous computational processes in artificial systems are 'forever' processes in operating systems. Operating systems carry out computational processes such as resource management and process control. For example, garbage collection. They are always live systems which return to the same ground state, and never provide a final output. The CTT does not cover a system that keeps going such as this, operating on finite data but in a potentially forever process. Manna and Pnueli (1992) showed that non-terminating reactive processes like operating systems cannot be modelled by algorithms. Since TMs do not capture forever processes such as this, they and equivalent input-output computational formalisms provide poor models for cognitive systems. This is because cognitive systems in living organisms that learn and develop over many years are not adequately modelled as a series of very many discrete and independent input-output computations. Following our 'building blocks' analogy, input-output computations that occur in complex computational systems that carry out more sophisticated computational tasks that are just one component of these complex systems. Newell defined a Cognitive Architecture as: *"The fixed (or slowly varying) structure that forms the framework that forms for the immediate processes of cognitive performance and learning"* ((Newell, 1990, p. 11). Thus emphasising the requirement for continuity in explanations of the computational mind. However, the CTT is misintepreted to suggest that input-output computational formalisms produce all computational behaviour of interest to Cognitive Science.

2.3.4. Indeterminancy

True random computation is outside the scope of CTT. A TM cannot produce random numbers. A TM can only produce pseudo random numbers and this fundamental difference to real random numbers is enough to make the difference between security and insecurity in systems, and there have been actual attacks based on this distinction (Kelsey, Schneier, Wagner, & Hall, 1998). Facilities such as hardware-based random number generators cannot be shoehorned into the paradigm for which the CTT was formulated. The key point for Cognitive Science is that natural systems such as human brains are realised in a different way than input-output computation systems, like TMs, or 1950s batch job computers where input is given on a paper card. So cognitive architecture implemented in human brains possess a capability for indeterminancy in computation that cannot be provided by TMs.

2.3.5. Real time

In models of computation within the scope of the CTT 'time' is not the same as the familiar notion of physical time . Rather it is "sequential time" as the CTT does not take into account the physical time required for an individual step in a program (MacLennan, 2009). So the number of steps can be counted but this does not translate into a duration in real time. Sequential time is reasonable in the context of mathematics

aimed at understanding formal derivability or effective calculability. In other contexts, such as understanding real-world behaviour of the kind that theorists in Cognitive Science are attempting to explain, this assumption may not be reasonable.

Unlike Turing machines, real computers are equipped with a *real-time clock*, serving a multitude of purposes such as time-stamping and synchronisation. Regarding computational expressivity, the clock extends the possible behaviour of the machine in interesting ways. It is now possible to measure the time that has elapsed between interactions (for example, in order to recognise a "double-click"). We can imagine a range of different capabilities such as whether the computer is able to manipulate the clock's setting or even adjust its speed. Inclusion of real-time information is important for explaining many mental phenomena. One example is the binding problem in perceptual psychology. This considers how features of objects, events, and background are perceived as a single experience (Doumas, Holyoak, & Hummel, 2006; Doumas, Hummel, & Sandhofer, 2008; Petters, Hummel, Jüttner, Wakui, & Davidoff, 2017). Computational approaches to consciousness and attention therefore need to focus on temporal synchronisation - when parallel processes occur at the same moment in time. Real-time information is also important in modelling many other cognitives process such as decision making (Mazurek, Roitman, Ditterich, & Shadlen, 2003) and many other cognitive functions that require timing intervals TODO Taatgen.

2.3.6. Richer input

Clocks are only one kind of *real-world sensor* that the computational process can be connected to. Myriad other devices are available, from temperature sensors to GPS. Clearly, it would be utterly absurd to try to reduce the capabilities of such *physically enhanced* computers to mathematical functions from $\mathbb{N}$ to $\mathbb{N}$, or to try to prove that machines equipped with a camera, say, are somehow equivalent to ones with an accelerometer. Richer forms of input includes real number computation, continuous input, and input with infinite inputs (streams). An example of the increased power of real numbers in computation is provided by Siegelman and co-workers (Cabessa & Siegelman, 2014; Siegelman, 1995, 1999)[3]. As with the issue of indeterminancy discussed in section 2.3.4, human brains are realised in a way that input-output systems like TMs are not. Affording the inclusion of richer forms of input that can then be incorporated into more complex computational systems.

2.3.7. Intentional computation and self-reflection

Intentional computation is also outside the scope of the CTT. This is of particular relevance to the issue of how meaning might be produced in a running programs. Intentional computations are those that query the internal structure of their arguments. For example, in JAVA one cannot write a method from `int` to `int` that will analyse how it was called, differentiating, say, between the invocations `fun(7)` and `fun(3+4)`. It is possible in LISP but this does not mean that JAVA and LISP are not Turing equivalent in terms of the functions they can compute, it's just that the capability for intentional computation is outside the scope of the CTT. Intentional computation is not just concerned with simple arguments. First-order computability discusses which functions $\mathbb{N} \to \mathbb{N}$ (or mapping strings to strings, or mapping finite objects to finite objects) are computable. Higher-order computability considers which functions

[3] we do not follow Siegelman and co-workers in claiming that richer input can lead to super Turing computation. Our arguments in this paper do not rely on this claim

involving infinite objects (such as infinite strings, real numbers, and functions) are computable. In addition, discussing how to compute these functions. In a higher-order setting, intentional computational queries perform program analysis. This is beyond the expressive power of traditional term rewriting systems, such as lambda-calculus or combinatory logic, as they are extensional. For an in-depth discussion see Jay (2019).

Formalisms that allow analysis of themselves, such as intentional computation systems, can create a kind of internal 'meaning' about their own system. A TM cannot carry out this kind of self-reflection. A computational formalism outside the scope of the CTT is needed to explain self-reflection.

2.3.8. Summary

We conclude that the Church-Turing Thesis is a fundamental, yet isolated phenomenon. It applies to input-output processes only. It has nothing to say about computation in the wider sense. Even within the tidy world of mathematical functions and input-output processes there is a well-known case where (provably even) an analogue of the CTT is not available. This is when we consider only those machines which *always* return an answer, or in other words, formalisms that are guaranteed to compute *total* functions from $\mathbb{N}$ to $\mathbb{N}$. There are many formalisms which have this (desirable) property, for example finite state automata or various typed lambda calculi. Adapting Cantor's diagonalization trick one shows that no such formalism can be expressive enough to cover all total computable functions, and indeed, there are provable differences between the calculi. Once again we stress that we are not postulating that "hypercomputation" occurs in these more elaborate and embedded systems (which would mean that *functions* can be reliably computed by these systems that Turing machines can not), but the point is that general computation is a much richer phenomenon than that afforded by simple input-output processes. On the other hand, note that our concern is a long way from philosophical controversies over what constitutes computation ("Is the paper in front of me a Turing machine?"); instead, our point of view is very conventional, we assume that all data is digital and that the elementary processing steps by which this data is transformed may indeed be simulated by a Turing machine. *The complexity arises from the way these steps are sequenced in time, how systems interact, and how they obtain digital data from real-world quantities.* This complexity is *irreducible* and far removed from Hilbert's input-output processes, and may be particularly relevant to the concerns of Cognitive Science.

3. Landscape of computability

The landscape of computability presented in Figure 1 is a diagrammatic representation of our claims. It shows which formal computational systems are within and outside the scope of the CTT. Our principle claim is that we refute the existing idea that all computation has a boundary between what are computable functions and non-computable functions that is a clear and distinct boundary for all formalisms. Classically we talk about partial functions from $\mathbb{N}$ to $\mathbb{N}$. This corresponds to what we mean by input-output or batch computing. With the discovery of the CTT, functions split into computable and non-computable. So the horizontal dashed line on the left of the diagram in Figure 1 demarcates computable and non-computable. In Figure 1, we refer to the bottom left square as *terminating* computations from $\mathbb{N}$ to $\mathbb{N}$ (full functions). In this realm any given formalism is *incomplete* in that it can be shown

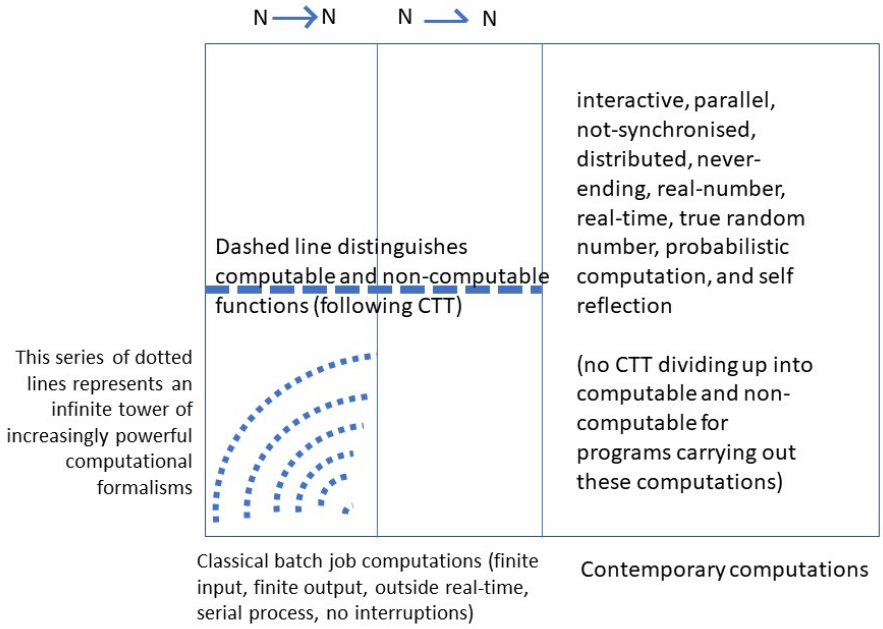

Figure 1.: The landscape of Computability showing the scope of CTT. There are two vertical lines in the diagram. The left line distinguishes between total and partial functions. The right hand vertical line distinguishes input-output computation from contemporary approaches based on general computation. The horizontal line through the left half of the figure separates computable from non-computable functions. Every reasonable formalism gives all lower left functions and no formalism gives any function in the upper left (according to the CTT). There is (as yet) no comparable thesis to the Church-Turing thesis for the right of the figure (general computation).

by diagonalization that from the formalism itself one can identify a computable total function not covered by it. Thus there is an infinite number of possible formalisms forming a tower of increasing computational power. So within the landscape of computability there are regions that show infinitely many formalisms of increasing power. The right hand side of the figure has no CTT. So there is no simple dichotomy that distinguishes the computable from the non-computable.

4. Considering widespread misconceptions about the CTT across different disciplines

This section serves to show that the misintepretation of the CTT we describe is widespread in Computer Science and in Cognitive Science and it has had a substantive influence on theoretical development in Cognitive Science. From within Theoretical Computer Science, evidence of how widespread misinterpretations of the CTT is is provided by Goldin and Wegner (2005), MacLennan (2009), and Schächter (1999). Sloman (2002) and Sloman (1996) observes the influence of misunderstandings of the CTT for Artificial Intelligence. Whilst Copeland (2017, section 2) points out that misunderstandings and misapplications of the CTT are typical of writing in Cognitive Science and related disciplines.

Like us, Wegner and co-workers emphasise the importance of interaction (Goldin, Smolka, Attie, & Sonderegger, 2004; Goldin & Wegner, 2005, 2008; Wegner, 1998). He explicitly shows that Turing machines cannot model machines with interactive input and output because interaction is not expressible by a finite initial input string (Wegner, 1998). Our treatment develops theirs by applying this finding to Cognitive Science and also emphasising the six other variants to TM processing in table 1 that lead a computational formalisms beyond the scope of the CTT

Sloman (2002) highlights that there were two independent strands of development leading to modern computers, that were eventually combined: systems for controlling physical mechanisms and operating on information developed from machines such as clocks, steam engine governors, weaving machines, Hollerith census processing machines, and many kinds of machines used in automated manufacture; and systems for performing abstract mathematical operations, which we reviewed in section 2.1.1. Using our terminology we state that only the latter are within the scope of the CTT. As Sloman (1996) notes, the CTT is not relevant to the former because: *"the formal concept of computation involves no notion of process, causation, or time, and is only concerned with structural properties"* (Sloman, 1996, p. 2). However, according to Sloman (1996) the CTT is frequently problematically invoked with systems for controlling physical mechanisms.

Considering our contribution against the contributions from Maclennan, Sloman, and Schachter, our contribution makes the further step of formalising a definition of the scope of the CTT and showing how formalisms within the scope of CTT can act as building blocks with other building blocks in creating more complex computational systems. For example, table 1 contributes to this literature by systematically setting out the seven classes of variations in TM processing that lead a formalism to be beyond the scope of the CTT.

From a different angle, Jay (2019) recounts how the CTT has been incorrectly invoked to rebut the discovery of calculi that are more expressive than the λ-calculus, for example, by supporting intensional functions. Jay recounts that when more expressive calculi such as the constructor calculus, pattern calculus, *SF* calculus, and

λSF were proposed, a common response was that the CTT shows that the λ-calculus can express everything that a TM can implement. However, according to Jay, this mistaken response is due to not appreciating that differences in expressive power can exist between calculi Jay (2019, p. 77)

In addition to misunderstandings around the frame of relevance and scope for the CTT, we believe even within its frame of relevance and scope there are subtle misunderstandings. For instance, it is possible to have a reimplementation of one language by another, but we argue that it is misleading to invoke the CTT to say *"all languages are equivalent"*. Interpretating or reimplementating one language by another doesn't add any more understanding than the concept of abstraction which occurs when high level languages are implemented in low level languages. See Felleisen (1991) for an in-depth exploration of this point.

Copeland (2017, section 2) points out that *"pernicious misunderstandings"* about how the CTT should be applied are typical of writing in Cognitive Science and related disciplines. According to Copeland (2017), contributors in these fields typically overapply the CTT to claim it demonstrates a *"treatment of the limits of mechanism"* (Copeland, 2017, section 2). So misunderstanding its fundamental yet limited correct interpretation. Our approach develops that of Copeland (2017) as he confines his discussion to input-output computation and 't consider the variations we set out in table 1.

Pylyshyn provides an example of a misinterpretation of the CTT in Cognitive Science that illustrates why the CTT has been viewed as underpinning the whole motivation for the computational theory of mind:

> *"The work of Turing, in a sense, marked the beginnings of cognitive activity from an abstract point of view, divorced in principle from both biological and phenomenological foundations. It provides a reference point for the scientific ideal of a mechanistic process which could be understood without raising the spectre of vital forces or elusive homunculi but which at the same time was sufficiently rich to cover every conceivable formal notion of mechanism (that the Turing formulation does cover all such notions is, of course, not provable but is has stood all attempts to find exceptions. The belief that it does cover all possible cases of mechanism has become known as the Church-Turing thesis). It would be difficult to overestimate the importance of this development for psychology. It represents the emergence of a new level of analysis, which is independent of physics, yet is mechanistic in spirit. It makes possible a science of structure and function divorced from material substance, while at the same time it avoids the retreat to behavioralistic peripheralism. It speaks the language of mental structures and of internal processes, thus lending itself to answering questions traditionally posed by psychologists"* (Pylyshyn, 1979, p. 24-25)

When Pylyshyn refers to mechanistic accounts for computation that are *"independent of physics"* and *"divorced from material substance"* he is referring to a state of affairs described by the concept of multiple realisability. This is when a given mental property, state or event can be realized with many different physical implementations. Multiple realizability is a critical concept in Cognitive Science and Artificial Intelligence. If multiple realizability is found to be true for the set of mechanisms underlying thought it means that thought is not tied to any one specific material implementation. As Piccinini (2015) notes: *"any multiply realizable property places structural constraints on any mechanism that fulfils it"*. (Piccinini, 2015, p. 122). We argue that the CTT has been interpreted within Cognitive Science in a way that doesn't adequately acknowledge the different nature of structural constraints that are required for multiple realizability of mechanisms that respectively support input-output computation and general computation. In addition, as we see from the quote above, Pylyshyn also

misinterprets the CTT to claim that the CTT covers all possible cases of mechanism. The distinction we previously highlighted between input-output computation and general computation dis-entangles mutliple realization from problematic reductionism. We argue that Cognitive Science can maintain a functionalist approach supported by a cogent example of multiple realization without that example promoting a problematic reductionism that TMs can do anything that any other computational formalism can do. So our approach maintains the strong motivation for the computational theory of mind and shows that this motivation does not need to be underpinned by a misinterpretation of CTT.

5. Benefits for Cognitive Science in refuting a reductionist interpretation of the CTT

In contemporary philosophy of mind it is usual to formulate functionalism in terms of causal-functional roles (Kim, 2011). Machine State Functionalism is an approach to functionalism developed by Putnam (1967) where the functional relations are computational relations specifiable by a Turing machine table. In this approach, mental states are Turing machine table states. As Kim explains:

> *"It is not merely that anything with mentality has an appropriate machine description: machine functionalism makes the stronger claim that its having a machine description of an appropriate kind is* **constitutive** *of its mentality. This is a philosophical thesis about the nature of mentality: Mentality, or having a mind, consists in being a physical computer that realizes a Turing machine of appropriate complexity and powers. What makes us creatures with mentality, therefore, is the fact that we are Turing machines."* (Kim, 2011, p. 151)

However, one of the reasons that Machine State Functionalism became less popular and was superseded as a variant of functionalism by causal-relation functionalism is that a TM can only be in one state at a time (Kim, 2011; Maley & Piccinini, 2013). As we can see from 1, TMs don't just fail to support more than one state at a time but also do not support interaction, real time computation etc. TMs were replaced as a model of mentality with a theory that was not explicitly computational. It didn't refer to machines states. Instead specifying functional relations between inputs, outputs, and mental states (Block & Fodor, 1972; Maley & Piccinini, 2013). The anti-reductionist thesis that we have proposed in this paper is therefore relevant to the demise of Machine State Functionalism. As whilst TMs cannot support concurrency other more expressive computational formalisms can.

Computational implementation can be explained by mapping accounts (Sprevak, 2019). Some mapping accounts assert that a sufficient condition for computational implementation is the existence of an isomorphism between a physical system's states and transitions, and the abstract computational system's states and transitions. Other accounts of implementation view an isomorphic mapping as a necessary but not sufficient condition. Isomorphic mapping accounts of computational implementation explain why computations are multiply realizable (Sprevak, 2019). As systems based on physical substrates such as silicon transistors, vacuum tubes, mechanical calculators based on 'clockwork', neurons, and perhaps other substrates, can implement computations where their physical activities can be isomorphic to the abstract structure for that computation (Sprevak, 2019). Sprevak (2019) examines triviality arguments against computational implementation that arise from explaining computational im-

plementation in terms of isomorphic mappings:

> *"Triviality arguments seek to show that computational implementation in physical systems is trivial in some worrisome way. A triviality argument might show that a theory of implementation attributes computation to too many physical systems, or that it attributes too many computations to those physical systems that compute, or both. Triviality arguments threaten to make trouble both for computational functionalism (the metaphysical claim that implementing a certain computation is sufficient for having a certain mental state or process) and for computational explanation in science (the scientific practice of explaining mental and behavioural phenomena in terms of physical computations)" (Sprevak, 2019, p. 175)"*

Our thesis is relevant to evaluating triviality arguments about computational implementation because sufficiency of complexity for a physical substrate in a mapping relation is relative to the complexity of the abstract computation. If we take an anti-reductionist approach to computation then we might have a hierarchical taxonomy of different physical implementing substrates that vary according to the complexity that would be needed for isomorphic mapping to different computational classes. For example, a computational formalism that included the 7 variations presented in table 1 may be significantly more complex that a TM or 'two counter' machine.

The same issue arises when TMs are invoked in computational theories of consciousness (Klein, 2019): for an anti-reductionist approach to computational consciousness a theory is not confined to invoking a TM. Rather, an anti-reductionist approach to computational consciousness may select from a hierarchical taxonomy of different computational classes with elements and capacities presented in table 1. In the introduction we made a building blocks analogy - input-output computation is just one of the building blocks for a complex computational system. These complex systems can be organised in more complex ways through interaction with the environment, concurrency, and in 'forever' processes, and alongside other components like real-time computation, richer input, indeterminancy, and self-reflection. In a hierarchical taxonomy only some formalisms higher in the hierarchy may be expected to support consciousness. The key point we make is that rejecting the reductionist interpretation of the CTT means we can propose computational accounts for consciousness without this leading to TMs or even simpler Turing complete formalisms like two-counter machines having to be able to be conscious. As our approach leaves open the possibility that consciousness arises from computation due to the conjunction of factors presented in table 1 in more expressive computational systems.

In his paper *"Minds, Brains, and Programs"*, Searle (1980), John Searle made an argument based on a "Chinese Room". This has a person inside ("Searle-in-the-room") who does not understand Chinese and who follows English instructions to process Chinese symbols according to rules without understanding the meaning of the Chinese symbols. Unbeknownst to Searle-in-the-room the outcome of his symbol processing is a pattern of actions which appear to be appropriately meaningful responses to questions posed to the Chinese room from the outside. So Searle-in-the-room does not understand what is processed (nor that it might have a meaning that might be uncovered by careful analysis) but the behaviour that results seems to be the result of understanding. Searle's lesson is that an observer external to the room would see meaningful behaviour but within the room there is only meaningless symbol processing — so according to Searle demonstrating that understanding cannot arise from just the operation of formal syntactic processes. Therefore, the CRA is a thought experiment that is intended to show that running programs (all running programs!) cannot have

understanding and awareness of what they are doing. It is this claim of generalisation to all programs that is a critical focus of the present paper.

In section 2.3.7 we showed that self-reflection can occur in general computation. This is because intentional computation in a higher order setting leads to program analysis. A possible but incorrect rebuttal of a new CRA scenario that showed how self-reflection gives rise to meaning would be to say: *"the self-reflection in that program could be broken down into smaller components, and hence implemented in a simple computational formalism like a TM — which certainly cannot self-reflect"*. However, this kind of reductionism is only definitely warranted within the scope of the CTT. Outside the scope of the CTT more expressive or powerful formalisms may not reduce to simpler formalisms. Self-reflective elements of a program that carry out program analysis may be derived from atomic/irreducible components of that computational formalism's implementation. More expressive or powerful formalisms may require particular implementing hardware to support more complex computational processes that are basic to that formalism. This does not mean that such formalisms can carry out function computations that input-output computation cannot carry out. It is, of course, possible to use a rich interactive machine to implement a simple function; after all, that is what we do with our modern computers all the time. It is our belief that this does not lead to new computable functions, i.e., some sort of "hypercomputation". In other words, the limits with regards to input-output computation established by the CTT are valid even if more sophisticated machinery is employed. It is the other direction that is the core of this paper: When considering more sophisticated computational tasks, then standard Turing machines (and their mode of operation) are not sufficient to explore the range of possibilities

6. Conclusion

Our main argument overall in this paper is that the way that the CTT is mistakenly interpreted leads to the view that a TM can do everything that any other computational formalism can carry out. Since we are not suggesting a form of hypercomputation, we are not suggesting that other computational formalisms can compute functions that TMs cannot compute. However, we claim that the more sophisticated computational processing carried out by systems that have computational capabilities beyond TMs does lead to greater computational expressivity. These differences in computational expressivity are relevant to both Cognitive Science and to Computer Science.

Turing set out to solve the "Entscheidungsproblem" (decision problem) and for this purpose proposed a mathematical formalism that faithfully emulates the process of a human being following finitely specified instructions. It was soon found that other formalisms have the same expressive power in this specific setting, i.e., mathematical problem solving, and this then led to the CTT. Since its first formulation in the 1940s no-one has really questioned the CTT, and nor do we. However, we emphasise its constrained setting: The CTT is concerned only with functions from strings to strings — input needs to be given as a fixed finite string and output (if it is produced) will be a finite string. The (partial) functions from strings to strings that can be computed by Turing's "machines" are the same ones that can be computed by any and all formalisms that have to date been put forward as alternatives. Put another way, the evidence for the CTT is very strong indeed, but this does not give licence to applying it — by analogy, as it were — to other forms of computation. These "other forms of computation" are not the fruits of idle speculation but very much day-to-day

reality for software engineers and computer users alike: computing machines no longer expect a question on a tape (or punch cards), go away and compute, and return an answer as a single file (or more punch cards or some output paper). Instead, contemporary computing also includes interactive, concurrent, continuous, real-time and indeterminate computation, with richer input, and intentionality and self-reflection. A canonical, maximally expressive formalism for computational processes simply does not exist. We have shown the CTT as it only applies to input-output computation is a fundamental yet isolated phenomenon. We point the interested reader to Samson Abramsky's account (Abramsky, 2014) where this fact is highlighted and explored. As he summarises: *"Finding an adequate characterization of informatic processes in general can plausibly be considered to be a much harder problem than that of characterizing the notion of computable set or function."* (Abramsky, 2014, p. 3).

In addition to critiquing how the CTT has been interpreted, this paper has highlighted an important link between the CTT and theories, concepts and approaches in Cognitive Science. We described the link between multiple realization and problematic reductionism and suggested these issues should be untangled. We argued that an incorrect interpretation of the CTT underlies rejection of Machine State Functionalism and underlies the persuasiveness of Triviality Arguments about Computational Implementation. Rejection of a reductionist interpretation of the CTT releases computational accounts of consciousness from explaining how TMs might give rise to consciousness. We also argued that an incorrect interpretation of the CTT underlies the persuasiveness of the CRA. Searle invokes the CTT in the CRA to say: "the limitations of the program in the CRA applies to all programs because of the CTT." Accepting this generalisation strategy, as Searle does, means there can be no syntactic formal processes in a program, of even fiendish complexity or strangeness, that will ever give rise to any kind of semantics. If, on the other hand, Searle cannot invoke the CTT for the CRA then whatever lessons he draws from the CRA only apply to the specific scenario he presents. We show that making generalisations from simple systems to all possible systems is therefore an unwarranted extreme reductionism. Instead, CRA scenarios that invoke computational formalisms need to be validated on a case-by-case basis for prospective program formalisms, leaving open the possibility that analysis of some other program formalism could lead to different conclusions.

Critically, for richer kinds of computation, some of which are described in this paper, the empirical evidence suggests that there are many shades of expressivity, which is why no-one has ever postulated an analogue of the CTT for them. Therefore this paper makes a radical contribution which attempts to overturn 80 years of misguided extrapolation, namely, that accepting the CTT implies that all programs that are of interest to computer science, cognitive science, and philosophy are nothing else but input-output processes. We also note that the computational variants we listed above (in table 1, are not only more similar to typical human cognition than the very simplified portrayal of processing of a TM, but the complexity they present is fast being achieved and overtaken in contemporary computing systems.

References

Abramsky, S. (2014). Intensionality, definability and computation. In A. Baltag & S. Smets (Eds.), *Johan van Benthem on Logic and Information Dynamics. Vol. 5 of Outstanding Contributions to Logic* (pp. 121–142). Springer Verlag.

Anderson, J. (2009). *How can the human mind occur in the physical universe?* New York:

OUP.
Bell, C., & Newell, A. (1971). *Computer structures: readings and examples.* New York: McGraw-Hill Book Company.
Block, N., & Fodor, J. (1972). What psychological states are not. *The Philosophical Review*, *81(2)*, 159–181.
Cabessa, J., & Siegelman, H. (2014). The super-turing computational power of plastic recurrent neural networks. *International journal of neural systems*, *24*, 450029–450051.
Clark, A. (1991). *Microcognition: Philosophy, cognitive science, and parallel distributed processing.* Boston: MIT Press.
Clark, A. (2014). *Mindware.* Oxford: Oxford University Press,. (2nd edition)
Copeland, B. (2002). Hypercomputation. *Minds and Machines*, *12*, 461–502. (2)
Copeland, B. (2017). Church Turing Thesis. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University. `https://plato.stanford.edu/entries/church-turing`.
Doumas, L., Holyoak, K., & Hummel, J. (2006). The problems of using associations to carry binding information. *Behavioral and Brain Sciences*, *29*, 74–75.
Doumas, L., Hummel, J., & Sandhofer, C. (2008). A theory of the discovery and predication of relational concepts. *Psychological Review*, *115*, 1–43.
Felleisen, M. (1991). On the expressive power of programming languages. *Science of Computer Programming*, *17*, 35–75.
Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, *38*(1), 173–198.
Goldin, D., Smolka, S., Attie, P., & Sonderegger, E. (2004). Turing machines, transition systems, and interaction. *Information and Computation*, *194*, 101–128. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0890540104001257` (Special Issue Commemorating the 50th Birthday Anniversary of Paris C. Kanellakis)
Goldin, D., & Wegner, P. (2005). The Church-Turing thesis: Breaking the Myth. In B. S. Cooper, B. Löwe, & L. Torenvliet (Eds.), *New Computational Paradigms* (pp. 152–168). Springer Verlag.
Goldin, D., & Wegner, P. (2008). The interactive nature of computing: Refuting the strong church-turing thesis. *Minds and Machines*, *18*(1), 17–38.
Hoare, C. A. R. (1985). *Communicating Sequential Processes.* Prentice Hall International.
Jay, B. (2019). Intensional computation with higher-order functions. *Theoretical Computer Science*, *768*, 76–90.
Kelsey, J., Schneier, D., Wagner, D., & Hall, C. (1998). Cryptanalytic attacks on pseudorandom number generators. In *Fast Software Encryption, Fifth International Workshop* (pp. 168–188). Springer-Verlag.
Kim, J. (2011). *Philosophy of mind, 3rd ed.* Boulder, Colo.: Westview Press.
Klein, C. (2019). Computation, consciousness, and computation and consciousness. In M. Sprevak & M. Colombo (Eds.), *The routledge handbook of the computational mind* (pp. 297–309). Routledge.
MacLennan, B. A. (2009). Super-Turing or non-Turing? Extending the concept of computation. *International Journal of Unconventional Computing*, *5*, 369–387.
Maley, C., & Piccinini, G. (2013). Get the latest upgrade: Functionalism 6.3.1. *Philosophia Scientae*, *17*(2), 135–149.
Manna, Z., & Pnueli, A. (1992). *The temporal logic of reactive and concurrent systems, springer.* Berlin: Springer.
Mazurek, M., Roitman, J., Ditterich, J., & Shadlen, M. (2003). A role for neural integrators in perceptual decision making. *Cerebral Cortex*, *13*, 1257–69.
Milner, R. (1980). *A Calculus for Communicating Systems.* Springer Verlag.
Monsell, S. (2003). Task switching. *Trends in Cognitive Science*, *7*, 134–140.
Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.
Petters, D., Hummel, J., Jüttner, M., Wakui, E., & Davidoff, J. (2017). Using computational models of object recognition to investigate representational change through development.

In G. Dodig-Crnkovic & R. Giovagnoli (Eds.), *Representation and reality in humans, other living organisms and intelligent machines* (pp. 141–173). Cham: Springer International Publishing.

Petzold, C. (2008). *The annotated turing: A guided tour through alan turing's historic paper on computability and the turing machine.* Indianapolis, IN: John Wiley and Sons.

Piccinini, G. (2015). *Physical computation: A mechanistic account.* Oxford, GB: Oxford University Press UK.

Preston, J. (2003a). Introduction. In J. Preston (Ed.), *Views into the Chinese Room: New Essays on Searle and Artificial Intelligence* (pp. 1–50). Oxford: Oxford University Press.

Preston, J. (2003b). *Views into the Chinese Room: New Essays on Searle and Artificial Intelligence* (J. Preston, Ed.). Oxford: Oxford University Press.

Putnam, H. (1967). Psychological predicates. In W. H. Capitan & D. D. Merrill (Eds.), *Art, mind, and religion* (pp. 37–48). University of Pittsburgh Press.

Pylyshyn, Z. (1979). Complexity and the study of artificial and human intelligence. In M. Ringle (Ed.), *Philosophical perspectives in artificial intelligence.* Humanities Press.

Schächter, V. (1999). How does concurrency extend the paradigm of computation? *The Monist*, *82*, 37–57.

Schlesinger, M., & Parisi, D. (2001). The agent-based approach: A new direction for computational models of development. *Developmental Review*, *21*, 121–146.

Searle, J. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, *3*, 417–424.

Siegelman, H. (1995). Computation beyond the turing limit. *Science*, *268*, 545–8.

Siegelman, H. (1999). Stochastic analog networks and computational complexity. *Journal of Complexity*, *15*, 451–475.

Sloman, A. (1996). Beyond turing equivalence. In P. Millican & A. Clark (Eds.), *Machines and thought: The legacy of alan turing (vol i)* (pp. 179–219). Oxford: The Clarendon Press. ((Presented at Turing90 Colloquium, Sussex University, April 1990. Also Cognitive Science technical report: CSRP-95-7))

Sloman, A. (2002). The Irrelevance of Turing Machines to AI. In M. Scheutz (Ed.), *Computationalism: New Directions.* Cambridge, MA: MIT Press. ((Available at http://www.cs.bham.ac.uk/research/cogaff/))

Sprevak, M. (2019). Triviality arguments about computational implementation. In M. Sprevak & M. Colombo (Eds.), *The routledge handbook of the computational mind* (pp. 175–191). Routledge.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *42*, 230–265.

Wegner, P. (1998). Interactive foundations of computing. *Theoretical Computer Science*, *192*, 315–351.

Wikipedia Contributors. (2022). Mathematics. In *Wikipedia.* `https://en.wikipedia.org/wiki/Mathematics`.