Relativity of Computational Descriptions

Piotr Jablonski [1]

Abstract. Computational Theory of Mind (CTM) claims that mental processes and states are grounded in, or even identical to, the computational processes and states of the human brain. As a general paradigm it occupies a middle ground between physicalism, which equates mental states with the physical states of the brain, and behaviourism, which proposes that only input-output dependencies, manifested by the agent in his behavioural reactions to the stimuli, matter. The opponents of CTM, notably represented by Searle and Putnam, argued that computational formalisms are not intrinsic properties of the physical systems but merely observer-relative ascriptions that map abstract formal structures onto the physical world in an arbitrary way. These arguments focus on the notion of 'realisation', i.e. the mapping from the abstract formalism to the physical world. I will argue that the computational explanation is not a single abstract algorithm, but rather a hierarchy of implementations with the various levels of detail accounted for. Computationalism remains observer-dependent because it fails to distinguish a privileged level of formal description that ought to be used as a basis for mentality.

1 INTRODUCTION

Advocates of the CTM propose that mental properties of cognizing agents are based on the fact that the brain is a computing hardware, realising a particular algorithm. Mental states are grounded in computational states, independent of the underlying hardware or a 'wetware' in the case of organic neural systems.

CTM tries to find a middle ground between behaviourism that ignores any notion of internal mental states and limits the understanding of cognitive processes to merely behavioural reactions to the environmental stimuli, and physicalism, which identifies mental states with the physical states of the brain and equates cognitive processes with the physiological activity of neuronal assemblies. The differences between these three approaches become clear when we ask what conditions have to be fulfilled by the two agents so we could determine that they possess identical mental states. Proponents of behaviourism would be satisfied if the two agents respond identically to the same stimulus. For a physicalist, they have to possess an identical physical composition of their brains. An advocate of the CTM would demand that they both have to realise the same computations in their brains. It is not however, clear what properties the two physical systems must possess in order to realise the same algorithm.

According to the CTM, details of human neurobiology, physiology or biochemistry, while helpful for understanding of the computational processes, are not essential to the cognitive or mental phenomena happening in our heads.

Classically defined computations as proposed by Turing [1] are formally defined, abstract transformations of symbols performed by equally abstract machines. Turing Machines (TM), Finite State Automata or any other equivalent formalizations are mathematical abstractions that do not exist in the physical domain. Moreover, they may possess qualities that cannot be realised by any physical system, e.g. TM require an infinite storage capacity, they are immune to errors in the execution of the algorithm etc. The physically realised computers are always a crude approximation of the ideal 'blueprint' that exists in the domain of the mathematical objects.

Whenever we claim that a physical system performs a computation, we ought to show a mapping that, within the constraints of the physical medium, will assign the appropriate physical states to the computational states of the ideal machine and map formal transformations of symbols onto the physical processes occurring in the artefact. However, the physical nature of the system remains irrelevant as long as the physical realisation remains structurally isomorphic to the abstract formalism [2].

As a result, every algorithm is *multiply realisable* i.e. it can be instantiated in many physically different arrangements of matter and energy. The same algorithm may be realised by the modern electronic computer, a mechanical system of gears and levers or a human being rigorously executing prescribed instructions.

This leads to the conclusion that a machine can exhibit mental states that are identical to human, as long as the computations realised by the computer are the same as the algorithm instantiated in the human brain.

But what does it mean that the two physical systems perform identical computations? I will argue that the answer to this question is, to a degree, subjective and can vary from the requirement for the identity at the most basic, 'hardware level' of description to the most general identity of functional input-output dependencies for the system.

While this argument does not negate the existence of computational levels of description, stuck in between the physical implementation and behavioural function, it shows that there is no single, privileged formalism that may serve as a basis for mental properties. Although I will not claim that we can make computational ascriptions ad-hoc in a way traditional opponents of the CTM suggest, nevertheless this relativism poses a serious problem to computationalism as a legitimate paradigm for cognitive sciences.

[1] Institute for Language, Cognition and Computation, School of Informatics, Univ. of Edinburgh, EH8 9AD, UK. Email: piotr.jablons@gmail.com

2 TRIVIALIZATION ARGUMENTS

Searle and Putnam [3][4] criticized CTM on the fact that computation per se is not a physical process or a phenomenon that can be observed in nature. It is not an intrinsic property of physical systems but merely an idealized description of the actual process. Their critique was aimed at the notion of 'realization', a mapping between abstract, formal procedure and the physical phenomenon.

Trivialisation arguments try to demonstrate that such mapping can be defined for:

A) an arbitrary algorithm and arbitrary physical process

B) an arbitrary algorithm and some, or a class of physical processes

C) some physical process and more than one algorithm.

Searle famously argued that the wall behind his desk may be interpreted as a realisation of the 'WordStar' word processing program. His vague remarks may be interpreted as a version of the Lycan's 'bucket argument'[5]. In both cases, the main idea seems to be that every macroscopic object when analysed on the molecular level contains a vast number of elements (particles) that are in some relations with each other and those relations change and evolve over time. Since we can label any part or property of a physical system as a realisation of an abstract symbol, we would be able to pick and label some particles in a way that the structure and evolution of the selected molecular subsystem would be isomorphic to the given algorithm and its execution. The difference between Lycan's bucket and Searle's wall would be that while Searle claims that he would always be able to pick an appropriate arrangement of particles for an arbitrary algorithm, Lycan only states that the water molecules in a bucket could by chance form the fitting pattern.

Putnam offered a different argument based on the fact that an execution of every algorithm may be represented as a sequence of state transitions of the inputless Finite State Automaton (iFSA), and since physical phenomena are continuous in time, we can map such sequence on the temporally adjacent states of the physical system. As a result, under the chosen labelling, the temporal evolution of the physical system would be interpreted as a realisation of the sequence of the computational states, and thereby the execution of the algorithm.

Responding to the arguments above, Chalmers [6] argued, among other things, that representation of the algorithmic procedure as a sequence of monadic computational states is an oversimplification. According to him, any formalisation of cognitive processes should possess a rich internal structure comparable to TM or proposed by him a *combinatorial state automaton*. At each step of the algorithm, the state of the machine is not given by the single monad but rather by the complex pattern or at least a vector of internal 'substates'. Every legitimate realisation of such automaton has to label this rich internal structure onto the appropriate arrangement of matter and energy.

It is worth noticing that all participants of the discussion seem to agree that the formal, abstract structure that is realised in a physical medium is a well-defined entity. While Chalmers argued that the formalism of iFSA is not appropriate, he seems to agree that the formalisation of a cognitive process would be a single abstract structure under the condition that it had the appropriate, rich internal composition.

I will argue that when we look into the actual practise of cognitive science we will have to face the fact, that every cognitive process can be formalised in many different ways, and we have no tool to distinguish the one that is privileged and essential for the process.

3 LEVELS OF COMPUTATIONAL DESCRIPTION

Marr has verbalized the main paradigm of cognitive science in his book "Vision" [7] where he suggested that we should be interested in some "high level" computational description of the cognitive processes, more fine grained than behavioural but much more general than the neurobiological level.

He defined three levels of understanding of any computing system as:

"- Computational theory - What is the goal of computation, why is it appropriate, and what is the logic of the strategy by which it can be carried out?

- Representation and algorithm - How can this computational theory be implemented? In particular, what is the representation of the input and output, and what is the algorithm for the transformation?

- Hardware implementation - How can the representation and algorithm be realised physically?"

While the two lower levels seem well defined, we should examine the level of 'Computational theory' with scrutiny. We may understand the goal of the computation as an environmental problem that needs a solution. Alternatively we may think of the goal as a map from the sensory input to the behavioural output that is appropriate and beneficial for the agent. In other words, the goal of the computation is to produce the right kind of output for the given input. I would argue for the later since it fits the discussion that will follow.

The second part of the definition, however poses a problem. What is "the logic of the strategy" and, more importantly, how does it differ from the 'algorithm' at the second level? The intuition behind it seems to be taken from programming experience. A programmer may have a general idea, how the algorithm should work, but she has not figured out all the details of the program. The 'logic of the strategy' would be this kind of general idea. However in the context of the three levels of understanding this addition seems superfluous. All the information that can be regarded as "logic of the strategy" is already included in the lower level of the algorithmic description.

I suggest that we should limit the higher level of description to simple input-output dependencies, which will be equivalent to the behavioural or functional description of the cognitive process.

4 HIERARCHY OF ALGORITHMIC DESCRIPTIONS

Computational descriptions, at what Marr would call the level of 'Representation and algorithm', form, in fact, a hierarchy or rather a tangled mess of different formalisations. We can picture it as a hierarchy of idealized programming languages. On the very top we can describe any cognitive process or, in fact, any computation as a single input-output operation. This is the level

of behavioural or functional description. At this level we do not ask *how* something is computed but only *what* is computed.

When we start asking *how* something is computed we have to choose the programming language in which the answer will be given. By programming language I do not mean any specific language but rather a formal framework with syntax, basic data representations, symbols and operations. Every teacher of algorithmic methods faces such a dilemma. Even though the techniques discussed are very general and can be used in many programming environments, the presentation has to be made in some specific language. Usually we choose some high-level programming language with the appropriate set of basic data structures and operations to make the presentation clear and simple. Often we would choose to include some libraries that render the demonstration even briefer and focused on the presented techniques instead of, on the specifics of data structures implementation or subroutines utilised for the execution of some fairly well understood operations. In this sense the 'programming language' consists of every representation and method for which we know 'what' it computes but we do not care or know 'how' it is computed. In other words for every programming language the basic operations and data representations are defined in a functional way.

However, each primitive operation may be defined as a composite of many operations on the lower, more fine-grained level. Similarly, each representation can be defined as a composite of variables on the lower level.

This approach is well known in the IT industry where it serves as a basis for many programming techniques. High-level languages are developed so developers do not have to deal with the hardware specific details; programming libraries serve as a source of pre-made operations that can be used by the programmer without any insight into the internal implementation of the procedure. More general, the concept of the Application Programming Interface is based on exactly this idea. At the level of API a programmer is only interested in *what* the given method does, not in *how* it is done.

We should stress however, that such relativisation of the algorithm to the chosen programming language is not a merely pragmatic endeavour. There is no way to present an algorithm without some formal framework. We always have to accept a set of functionally defined basic blocks from which we will build the algorithm. We may choose a binary data representation, logical gates and memory registers as in the Von Neuman architecture or a tape, a set of symbols, internal states of the head and primitive read-write operations as in the TM formalism, or some high level programming language augmented with libraries like Java. We cannot, however present an algorithm outside of any such context.

Hardware implementation may be understood in two ways. It may be a physical description of the artefact that realises the algorithm. In this case it will be formed in the language of some scientific theory and does not belong to the computational description. On the other hand it may be a formal description of the algorithm in programming language which primitive operations and representations are mapped directly onto the physical processes and properties of the system. In the second case, hardware implementation is a special case of algorithmic description that is privileged by the fact that we are able to design and build the appropriate machine or by the biological architecture of the neural systems we happen to investigate. It is not, however some fundamentally basic level of description and we could always come up with another, even lower level of implementation.

The last point is nicely illustrated by the work of Paul Rendell [8] [9], who demonstrated how the TM can be implemented on the Conway's 'Game of Life' cellular automaton. Even though we tend to think about TM as a very basic structure, where every operation can be executed by an "idiot" (i.e. someone without any understaning of the computational process), we may always ask 'how' these operations are implemented and give the answer in the form of the detailed description of spatio-temporal pattems of the 'Game of Life' and the primitive operations governing the behaviour of the cellular automaton.

On the other hand, if the TM formalism was a description on the level of Hardware implementation, we would be unable to answer this question and could only point outside of the computational world, to the physical description of the actual machine.

In a sense, an algorithm is *multiply realizable* or rather *multiply implementable* long before we reach the physical domain.

Alternatively, one could hold the radical view, that at every level of implementation we deal with a separate algorithm. This view however, seems overly strict. First, it would mean that any physical realisation of a computing device realises many algorithms at once or only the one directly mappable onto the hardware (this however would mean that we could not claim that a high-level computational formalism akin to Fodor's Language of Thought [10] is realised by the hardware). The second problem with this position is that we cannot say that two slightly different implementations are still the same algorithm. E.g. we would have to conclude that two compilations of a program for two different hardware architectures are in fact, two distinct algorithms.

The question arises, what is the proper level of computational description of cognitive, and more general, mental processes. If we treat CTM as merely epistemic paradigm, a research method useful for cognitive sciences, we can argue that the level that gives the simplest description is the one that should be accounted for in the theories of cognitive science. However if one wants to defend the strong version of CTM, he has to present more fundamental reasons why any level of description should be privileged and why only it sustains mental states. For example, if an algorithm for visual recognition called for a sorting routine that returns some specific arrangement of data, we could ask if the way 'how' the sorting is implemented is important for the mental aspects of the process or not.

5 EXAMPLE

Marr gives the example of simple cognitive operation found in the flight control system of the housefly. The landing system realises a simple operation: if the image in the visual field of the fly 'explodes' fast enough as the fly approaches a surface, the fly automatically lands on the approaching surface.

The description given above is made on the behavioural level. It specifies only the input (image in the visual field), the output (the behaviour), and the function from input to output (if the image explodes - perform the landing behaviour).

It is hard to see what level of detail should be included into the 'computational theory' of this process on the Marr's original account.

Representation and algorithm may be defined on different levels. We may assume that visual field is represented as a primitive, monadic representation, a 'picture', the 'exploding' or rapid scaling-up of the picture can be detected by another primitive operation and the landing operation is the third primitive operation. Such an algorithm could be written in some very high-level, script language specialised in programming robotic houseflies.

On the lower level we may define the 'picture' as a binary matrix of black and white pixels. Detection of 'exploding' can be performed by comparing successive picture with the scaled up versions of the previous picture for the different scaling factors. If the successive picture matches the scaled-up version of the previous one, the visual field exploded with the velocity corresponding to the scaling factor.

On the, yet lower level we may define the scaling operation as a complex routine where the pixels are represented in polar coordinates and scaling takes form of multiplication of the distance of the pixel from the centre of the image.

Lower still, multiplication can be a composite of bit wise operations on the binary representations of the coordinates.

If we realize the given algorithm on the digital computer, at some level we would define the procedure in the form that is directly mappable onto the hardware, i.e. In terms of machine level commands governing the access to the memory and logical gates. It is worth noticing, however, that this 'hardware implementation' level of description is privileged only because we have chosen this specific hardware. In all probability, the 'hardware implementation' appropriate for the housefly neural system will be completely different. The picture would be represented as a pattern of neuronal activity, not a bitmap and scaling detection would probably take the form of a very different process.

The point here is that the two computational descriptions would split at some level. They agree at the top, functional or behavioural level, maybe at some lower levels but, eventually they drift apart. The problem for the computational account is to determine whether the two implementations are computationally equivalent or not. They are behaviourally equivalent and physically distinct. However, depending on the choice of the computational level and the 'programming language' they will be either equivalent or distinct in the computational sense. Since we do not have any fundamental reasons for the choice of the computational level, the answer to this question remains subjective or observer-relative.

6 CONCLUSIONS

Computational descriptions form a hierarchy where only the top, behavioural level is privileged. Hardware level description is dictated by the choice of physical realization of the computation.

Every level of algorithmic description is equally valid and there is no reason to select one that may serve as a basis of the mental properties as long as we want to keep multiple realizability in the picture and steer clearly between behavioural and hardware levels of description.

It is unclear what level of computational detail ought to be preserved in different physical realizations in order to 'generate' the same mental properties. In other words, at what point, going from hardware to behavioural levels, two different realizations should converge to the same description?

REFERENCES

[1] Turing, A., 'On Computable Numbers, with an Application to the Entscheidungsproblem', *Proceeding of the London Mathematical Society*, (series 2), 42, 230-265, (1936).

[2] Turing, A., 'Computing Machinery and Intelligence', *Mind*, 59: 433-460 (1950).

[3] Searle, J., 'Minds, Brains and Programs', *Behavioral and Brain Sciences* 3: 417-424, (1980).

[4] Putnam, H., *Representation and Reality*, MIT Press, (1988).

[5] Lycan, W., G. *Consciousness*, MIT Press. (1987).

[6] Chalmers, D. J., 'Does a Rock Implement Every Finite-State Automaton?', *Synthese*, 108, 309-333, (1996).

[7] Marr, D., *Vision*, CA: W. H. Freeman, (1982).

[8]Adamatzky, A., Durand-Lose, J., 'Collision-Based Computing', *Handbook of Natural Computing*, 1949-1978, (2012).

[9] http://rendell-attic.org/gol/tm.htm

[10] Fodor, J., *The Language of Thought*, Harvard University Press, (1975).