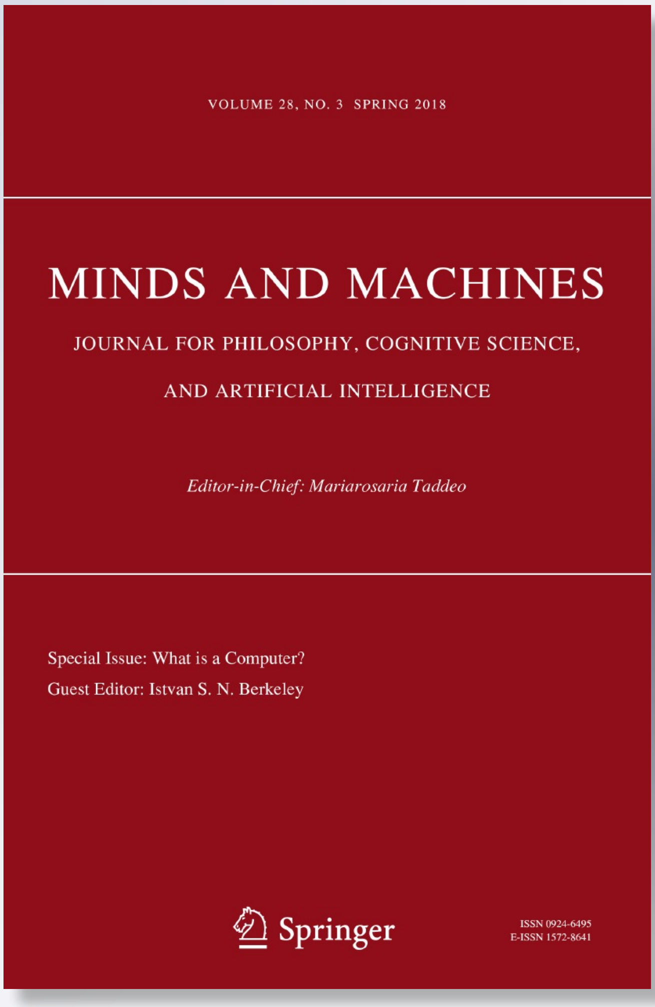
Computers Aren't Syntax All the Way Down or Content All the Way Up

Cem Bozşahin

Minds and Machines
Journal for Artificial Intelligence, Philosophy and Cognitive Science

ISSN 0924-6495
Volume 28
Number 3

Minds & Machines (2018) 28:543-567
DOI 10.1007/s11023-018-9469-2


VOLUME 28, NO. 3 SPRING 2018

MINDS AND MACHINES

JOURNAL FOR PHILOSOPHY, COGNITIVE SCIENCE, AND ARTIFICIAL INTELLIGENCE

Editor-in-Chief: Mariarosaria Taddeo

Special Issue: What is a Computer?
Guest Editor: Istvan S. N. Berkeley

Springer

ISSN 0924-6495
E-ISSN 1572-8641

Minds and Machines (2018) 28:543–567
https://doi.org/10.1007/s11023-018-9469-2

Computers Aren't Syntax All the Way Down or Content All the Way Up

Cem Bozşahin[1]

Received: 16 August 2017 / Accepted: 5 July 2018 / Published online: 18 July 2018

Abstract
This paper argues that the idea of a computer is unique. Calculators and analog computers are not different ideas about computers, and nature does not compute by itself. Computers, once clearly defined in all their terms and mechanisms, rather than enumerated by behavioral examples, can be more than instrumental tools in science, and more than source of analogies and taxonomies in philosophy. They can help us understand semantic content and its relation to form. This can be achieved because they have the potential to do more than calculators, which are computers that are designed not to learn. Today's computers are not designed to learn; rather, they are designed to support learning; therefore, any theory of content tested by computers that currently exist must be of an empirical, rather than a formal nature. If they are designed someday to learn, we will see a change in roles, requiring an empirical theory about the Turing architecture's content, using the primitives of learning machines. This way of thinking, which I call the intensional view of computers, avoids the problems of analogies between minds and computers. It focuses on the constitutive properties of computers, such as showing clearly how they can help us avoid the infinite regress in interpretation, and how we can clarify the terms of the suggested mechanisms to facilitate a useful debate. Within the intensional view, syntax and content in the context of computers become two ends of physically realizing correspondence problems in various domains.

Keywords Computer · Explanation · Semantic content · Learning computers · Correspondence problem

The paper is dedicated to my Arizona State classmate Jonathan Mills, 1952–2016, whose work led me to some rethinking which eventually led to this paper, who had taught me one day in class what it means to be a computer science graduate student.

✉ Cem Bozşahin
bozsahin@metu.edu.tr

[1] Cognitive Science Department, Middle East Technical University (ODTÜ), Ankara, Turkey

1 Introduction

There are many ways to realize the idea of a computer. Digital computers can be electronic or optical. Analog computers can conduct heat or electricity, use hydraulics, and manipulate air pressure for various tasks. For some computing enthusiasts, computing in nature is self-evident. Such variety might suggest that there are different types of computation because there are different kinds of computers.

I argue that there is one idea of a computer. Defining computers as calculating machines, and digital computers and analog computers as different types of computation, in fact defining any different realization of computers as something different in kind, is not very helpful in understanding what we can do with their capacity. We have to be clear about the terms and mechanisms of a computer, because these elements have to be physically realized to do the actual computing. Insisting on this principle allows us to clearly distinguish understanding the nature of a problem, understanding nature, and understanding the computational nature of a problem. Part I of the paper covers these issues.

We shall see that computers cannot be syntactic machines all the way down. They have to have non-syntactic primitives, both in their abstract form shown in first part of the paper and in their physical form shown in the second part to be able to carry out their syntactic processing.

The second part of the paper also argues that these results may or may not be good news for computationalism as currently conceived (i.e., cognition is computation), or for machine functionalism in general (i.e., a machine that functions as a mind counts as a mind). For example Searle, who is the main opponent of Strong [A]rtificial [I]ntelligence (the idea that computers can have human minds), was right to reject the Turing Test as a test for understanding, but as we shall see, in doing so he misrepresented the idea of a computer.[1] Turing (1950) seems to have made matters worse by unfortunate comparisons between machine and human intelligence without proposing an empirical theory for them. "Being suitably programmed" is not an empirical theory.

Adopting the computer as the basis of cognitive functions cannot be an automatic explanation of the mind as a computer. An empirical theory of content has to be provided using the computer, not appealing *only* to the core functions of the computer, but, crucially, reducible to their execution. My argument is that this theory of content, even if it satisfies everyone's criteria for adequacy, would only show that we can explain the mind, through a computer, no more no less. This concurs with the historical argument that it does not make the computer mindful. Then, anyone can take the intentional stance of Dennett (1971) if they wish, and treat the model as mindful, which is already happening in public conception of AI, even for dull programs.

All of these I believe are more or less good news because they can lead to new research programs involving computers, including social sciences. The bad news

[1] The test compares two physical medium-independent verbal exchanges with a machine and a human, and which is which is decided by a human questioner.

is that some objections to machine functionalism, as well as the support for it, can be shown to be based on a partial understanding of the computer. I suggest that the whole idea of machine functionalism begs many questions. The intensional view of the computer as the basis for theory-making may break the circle. The idea is based on the observations described below.

Given the logically possible landscapes of simple and complex behaviors related to content on the one hand and simple and complex mechanisms on the other for studying them scientifically, it is quite clear that targeting simple behavior with either simple or complex mechanisms is not very exciting, for we know that most of the interesting behavior worthy of studying is complex. Considering that in the current state of science, studying complex behavior with equally complex mechanisms is not very exciting either, we can narrow down the role of computer science in philosophy to making the terms of the computer, which is a simple mechanism, sufficiently clear so that we can hypothesize about content, including its uncertainty.

However, the term 'content' is overloaded. Even further clarifications such as 'semantic content' or 'broad vs narrow content' are not clear enough to put them to work. Roughly, broad content is the physical state of an object, with a causal history attached to the state, and narrow content is the current state of an object. It is better that we work on *how* things related to content can be studied. This allows us to concentrate on the correspondence problem of content and execution by making its terms clear.

We shall see that calculators fall far short of that goal. Defining the computer as a "calculator with large capacity" and the calculator as "a mechanism whose function is to perform a few computational operations on inputs of bounded but non-trivial size", as promoted by Piccinini (2008):33, does not help us understand what it is that we can do with that capacity.

The intensional view of computers promoted here is the view that the mechanism of the computer, if adequately defined at all levels, from its ontology to its processes, rather than enumerated through examples or behaviors, provides an opportunity for understanding a problem, but does not by itself count as an explanation. I think it lays at the heart of using computer science for explanations, as model-building *for* something, using computers, to explain it. It seems to create cross-purposes with the nomological view of computers in the philosophy promoted by Shagrir (1999), Copeland (2002), Piccinini (2008). For example, the intensional view differs from Shagrir's (1999):146 definition of [C]omputer [S]cience in which "CS is, indeed, the science of computers, whereas computers are physical dynamics (either [sic] algorithmic or not) that are individuated by the formal content of the representations over which they are defined." According to that view, content lies at the heart of computer science, as Shagrir later proposes in the paper. According to the intensional view, computer science allows us to hypothesize about content without infinite regress and with clear abstract and physical connections of *what* and *how*. It is not about content *per se*. It requires being very clear about the terms of the mechanism if we want a computational explanation. This in return requires being very transparent about computation in all its stages. In the end what is explained is not computers but the empirical phenomenon modeled by them.

Even the Church paradigm of computing, the lambda calculus, is too extensional to serve as a mechanical explanation in this sense. Its behavioral equivalence to Turing machines (to be defined in the beginning of Sect. 2) is sometimes not enough. We look for more expressibility. For example, there is a Turing-computable function that distinguishes the two lambda-calculus functions symbolized as *SKK* and *SKS*,[2] but the difference cannot be represented by the *SK*-system which gives us the Turing equivalence of lambda-calculus by writing all lambda terms with *S* and *K*, because *SKK* and *SKS* both yield the behavior of the identity function (Jay and Given-Wilson 2011). Such functions are Turing-computable intensional functions that are not representable by *SK*.

The implication is that there is more to computing than what is offered by the answer to the behavioral question, for example inspecting its own structure. That is why we commonly define computer science as answering the question "what can be automated?", knowing that the ensuing *how* question is addressable because of the Turing Machine (TM), being realized in some physical way, digital or analog.

There are also correlates of intensionalism versus extensionalism in the programming paradigms too. All programming languages worthy of the name are Turing-complete; that is, any algorithm that can be input to a Turing machine can be programmed in such languages. But they are not equally expressive. For example, although many languages provide macros, which are fragments of code to be inserted into other code from a template, in languages like Common Lisp we can write macros with which we can write programs that write programs, and before running the result we can inspect it by programs, which are, in this case, macros. The elusive intensional function as described above seems to be just such a macro-behaving function as suggested by Marc Hamann in 2010, in a blog discussion of Jay and Given-Wilson (2011) initiated by the authors.

Graham (1994) and Hoyte (2008) explained what we gain from this way of programming. This is the so-called 'programmable programming languages' paradigm in programming language design. Common Lisp macros are *compile-time* functions that use the same instruction set as run-time functions.[3] Infinite expansion of recursive macros to infinite forms is avoided because macro expansion itself is a built-in function. There are lessons to be learned here on infinite regress and physics of computing, from computer science to philosophy, in support of Dennett's (1991) Cartesian Theater warning.

The rest of this paper is organized as follows: Part I attempts to clarify the mechanism of the computer and promotes a single idea of computing. It also addresses potential contributions of the intensional view for using computer science in explanations, which requires a close look at correspondence problems. Part II argues that questions related to content require an empirical theory so that computers can be put

[2] The combinators *S* and *K* are respectively the lambda terms $\lambda x \lambda y \lambda z.xz(yz)$ and $\lambda x \lambda y.x$.

[3] A compiler is a program which translates from one programming language to another. Typically, the target language is a lower-level language than the source language. The final process in compiling is called *code generation*, by which all intermediate representations of levels of translations are dispensed with, and the code is solely in the form of the primitive instruction set of the intended architecture. There is no syntactic translation at run-time of compiled programs; it is just execution.

to use for solving *content's correspondence to form* without infinite regress. In this task assuming a computer as the underlying mechanism does not by itself count as an explanation. This way of thinking may avoid misunderstanding or misrepresenting the computer.

2 Part I: Computers and Their Mechanism

The best theory to date for computability, that of Turing (1936), is an abstract mathematical object in the form of an automaton, and even in the abstract form it is physically realizable because it has primitives which are not reduced to other operations, and whose terms are quite simple and clear: move left, move right, change state, read, and write. They need no external executive.

'Read' and 'write' instructions may need something external, but they are themselves not external for the Turing Machine. This can be seen clearly in the mathematical definition of a TM. Formally, it is $\mathrm{M} = (Q, \Sigma, \Gamma, \Delta, s, h)$, where Q is a finite set of states, Σ is the input alphabet, Γ is the tape alphabet (things that can be put in the infinite tape), and Δ is a finite set of transitions each of which must make reference to the primitives mentioned above and to Q, Σ, Γ. Without reference to the primitives, i.e. only with reference to Q, Σ and Γ the state of the computation cannot progress. For example, if we have a member of Δ in the form of $\Delta(p_1, a) = (p_2, b)$, it means that in state p_1, *reading a* from the tape and *writing b* on the same cell *end* in state p_2. These are not mere mentions of Q, Σ and Γ material but actions involving them.

In the remainder of the definition above, s is the start state, and h is the halt state in Q. This is the simplest conception of a TM as an abstract computer. Other variants, such as TMs as recognizers or transducers (i.e. generators) can be related to this definition; see Lewis and Papadimitriou (1998). I will use this source throughout the article as main reference for formal definitions.

2.1 Computers on Paper, in Blueprint, Digital or Analog

Today's *physical* computers have much fancier instruction sets. Nevertheless we know that anything we can write on paper using a Turing machine, we can write with their instruction set, and vice versa; otherwise they would not be called computers, at least not well-designed universal digital computers. They are also physically realized. We have managed to design circuits for them without violating the known laws of physics.

The scientific success and societal impact of the Turing paradigm are due to a surprising mixture of its mathematical beauty, which many mathematicians take to be its simplicity, and physical realizability. Showing the step-by-step progress of a function's concrete state in excruciating detail was uncommon. We were used to the global views of Frege and the lambda calculus of Church. This led to an understanding of quite complex tasks, and crucially, at the same time showing transparently that it happens without a concomitant increase or complication in the internal mechanism.

Here, we must distinguish the term *mechanism* from *architecture*. Architectures consist of real devices or blueprints for real devices, whereas a mechanism is an abstract object with some desirable properties such as physical realizability and transparency. Actions of the Turing machine are at the level of mechanisms; e.g., 'move to next space'. 'Next' and 'space' are abstract but physically realizable concepts. One can in fact build a physical machine with TM's primitives. In an architecture, they will have a physical correlate, using the representation relation described in the next section in footnote 4. This mapping is required because the physical evolution of computing, i.e., the computer's execution, is carried out at the physical level with physical objects; see Sect. 2.2.

The distinction is crucial so that we do not speak of the "speed of a Turing machine" because it is not an architecture but a mechanism, and as such it cannot have a physical property such as speed. We cannot consider its symbols as proxy representations of the mind either, as sometimes assumed in psychology on behalf of computationalism; e.g., in Bickhard's (1996) interactionism as a purportedly radical alternative to computationalism. Symbols only provide an opportunity to address the problem of reference, but they are themselves not empirical models of reference in a Turing machine.

Gandy (1980) makes the architecture-mechanism distinction very clear and suggests fundamental laws for any computing device to be called as such, which are expressed at the level of mechanisms. Copeland and Shagrir (2011) add a mid-level between the abstract mathematical object and the real device, what I called a blueprint above. Turing did not switch back and forth between architecture and mechanism when talking about the Turing machine, except perhaps in his discussion of Oracle machines, which are idealized and can make decisions instantaneously. They are abstract objects only, and of course they cannot be asked to justify the causal/physical chain of their steps.

As Copeland and Shagrir concede, although Turing referred to space, he made no reference to physical time. His space only depends on the notion of 'next' (next tape cell, next input, next state), which is an abstract term referring to a mechanism, not a physical term. Therefore, the difference between purist's view and realist's view of the Turing machine is really a question of daily practice when we wear different hats, rather than ontological commitments. In either case, at the mechanical or blueprint level, today's computers with random-access memories or abstract Turing machines with something less serial than a tape or one with more tapes cannot compute more problems than the set of Turing-computable functions; see Lewis and Papadimitriou (1998). This is why many of us think Turing's definition seems convincing in capturing the limits of computability.

The mixture of scientific clarity and realizability made programming available to a wide community. It seems to have captured some natural instincts, perhaps related to everyday planning, so that non-professionals and children also feel at ease programming a digital computer.

However, analog computers are programmable too, such as the extended analog computer of Rubel (1993), but currently its programming requires a large amount of

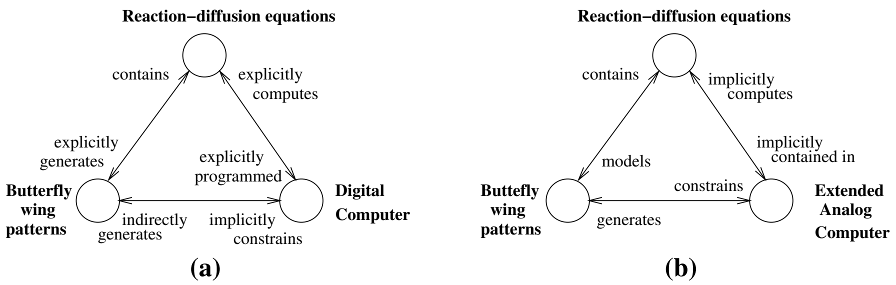


Fig. 1 **a** Digital and **b** analog computer paradigms, respectively giving **a** algorithms and **b** analogs for butterfly wing morphogenesis, from Mills (2008). Objects are in bold, relations are in normal font

scientific and engineering expertise. It uses the idea of fixing the functional structure of the problem by determining a *configuration* for the analog computer using mostly implicit functions drawn from nature. These are called *analogies* rather than algorithms (see Mills 2008). Figure 1 compares the two paradigms of computers.

2.2 Understanding the Nature of the Problem Versus Understanding Nature

Shagrir (1999) reported on a Pitowsky (1990) experiment, in which averaging three numbers is achieved in an analog way by a "thermal machine." The numbers k, m, n are thermometer's readings for the temperatures of the liquid, from three insulated containers separated by removable barriers. We remove the barriers simultaneously and wait until the temperatures equalize. The output is assumed to be $(k + m + n)/3$. Thus, thermodynamics can give an instant "computation" of the average for any number of containers, according to Pitowsky.

Shagrir asks us to question whether this computation is algorithmic by comparing it with the sequential removal of the barriers between the containers, which is assumed to be algorithmic. The more pressing question appears to be whether it is computation at all. We cannot repeat the experiment with gas, we cannot conduct it outside the earth's atmosphere, nor can we do it with non-conducting material. What we seem to show an understanding of through this procedure is the nature of thermodynamics, heat conductance, and living on earth, but not through computation.

What can be understood is averaging itself when we map these quantities to (say) numbers via a representation relation,[4] depending on the medium, then encode these numbers for the initial state of the physical computer using the inverse of the representation relation, then let the physical device calculate the evolution of its initial state to a final state (this is the Pitowsky stage above), decode the result back to the

[4] Take f to be the representation relation. It can be for example $f(\text{ low voltage }) = 0$, meaning we map the physical property of low voltage to bit 0. Decoding and encoding are different uses of a representation. Decoding is essentially using f. Encoding is using f^{-1}. For example, when we type 'a' in our word processor, we encode whatever physical property—some voltage levels—is assigned to it down below. Therefore good representations are needed technologically, both in digital and analog computers, to be sure about the physical component.

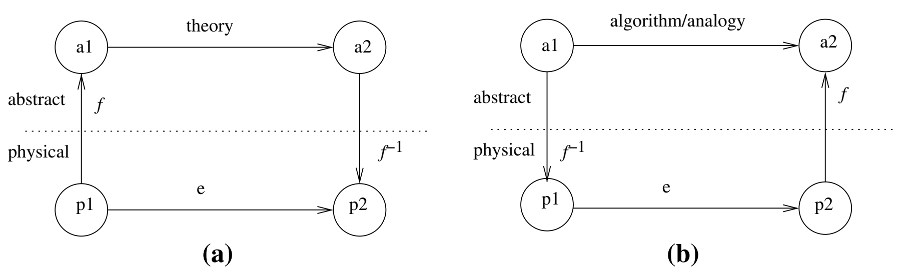


Fig. 2 The modeling relation (**b**), from computer science, and reversing the modeling relation (**a**), from physics, adapted from Horsman et al. (2013). 'e' is the evolution/progress of the physical system, 'a' is the abstract state, and 'p' is the physical state. *f* is the representation relation. The *use* of *f* is decoding, and f^{-1}, encoding. The end points of the commuting diagrams show whether we understand nature (**a**) or the nature of computing (**b**)

abstract domain of numbers, and see if *abstract* evolution of an averaging mechanism, captured either algorithmically (code) or analogically (equations), predicts the physical outcome.

This is the 'compute cycle' reported in Horsman et al. (2013); see Fig. 2.[5] In cases where it does not predict the expected outcome, we can see the value of computer science. Maybe (i) our theory of averaging was not good, or (ii) the compiler which translated our algorithm that we coded in a programming language mistranslated the algorithm, or (iii) the physical evolution did not amount to anything because the abstract primitives were translated to their physical counterparts erroneously, or (iv) the representation relation was useless, which means the encoding and decoding which depend on it would also be useless. If the computer and the compiler are well-designed (equivalently, if the implicit relations of the equations and the extended analog computer are well-designed), then the second to fourth cases would be known not to hold, so we go back and change our theory of averaging by revising our algorithm or equations.

It is difficult to see how, without decoding and encoding, that is, without modeling, using nature by itself, we could discover what went wrong. This is only possible if we are clear at every level about what pieces can do what, and how. It does not

[5] The authors note that the compute cycle is the reverse of the experiment cycle in for example physics. In physics we let the abstract level do its work, then encode its result in some physical object to see if the theory predicted its presence, location etc. correctly, as in Fig. 2a. The physical level does not do the work; its work is predicted. In the case of computers, the physical layer does the work—therefore it is not a simulation, and the result is tried to be predicted by an algorithm or an analogy (implicitly computes relation) of Fig. 1, as in Fig. 2b. Notice that what allows the physical layer to carry out its work is a series of translations of say the averaging algorithm or equations to the terms of the computer. We compare its physical work with the prediction at the abstract level, i.e. by an algorithm or equation.

Notice also that a computer scientist seeking an explanation has to reverse the modeling relation too, *after* it is established. Because it only serves to establish the correctness of the algorithm, assuming the underlying physical machinery was confirmed. What we do with the corrected algorithm afterwards is much like the diagram in Fig. 2a. Therefore physics and computer science may differ on how they use the model-theory-technology cycles, but it is clear that what amounts to theory in physics and computer science is based on the same process of thinking.

follow that the algorithm or the analog being tested must be deterministic. In fact it follows that any non-determinism ought to be part of the empirical theory and its models, as for example practiced in probabilistic machine learning, computational intelligence and evolutionary computing, because we know that non-deterministic Turing machines are stylistic variants of the standard Turing machine (see Lewis and Papadimitriou 1998, chapter 4). A non-deterministic Turing machine faces more than one choice of action in steps of the computation. We must be careful to look at all computations of the non-deterministic machine, which means that we must assume there is an upper bound on the number of actions depending on the machine *and* the input. The mechanism, however, is as clear as before. We can keep this theoretical result coming from the Turing machine in the back of our minds and continue to compute digital or analog way as we wish.

It is also important to realize that the modeling relation is not a simulation. If we take the term 'simulation' in its technical sense, which is behavior generation from a model, then it is clear that modeling also sets the causal relation for it. Models and simulations are different species. Therefore, for us to see whether a computation matches the abstract evolution predicted by an algorithm or an analog (say an equation), we have to use the model; that is, we physically undertake the computation.

Analog computers do not just leave the physically evolved result as such to finish their work. They need to translate back, referred to as *implicitly compute* in Mills (2008) and in Fig. 1, to the terms of the equations, for the result to be of any use. This is computation. Needless to say, the digital version is also computation because it is the common understanding of an algorithm. What lacks in the experiment in Pitowsky (1990) is the translation of the analog results to be of any value. The question is to what they translate into. Surely it is an abstract domain, whether it is represented as an algorithm or an equation. Therefore, the choice we are forced to make according to Shagrir (1999) about either leaving analog computers aside if we want to talk about computation today, or assume that everything in nature computes, is a false dichotomy.[6] I think that making the terms of computation clear offers an explanation for the difference in understanding a problem and understanding nature.

Now consider another problem in computer science: sorting. It is a subfield by itself in the study of algorithms, but it can also be investigated without algorithms. Dewdney (1984) proposed a method inspired by nature to sort the "analog way". It may be analog, but it is not clear whether it is a computational analog; i.e., an equation for an analog computer. We can take a sheaf of spaghetti, bang it on a table, and

[6] One implication of this result is that nothing computes in nature unless we map it to a computational problem in our thinking. Pancomputationalism and born-again computationalism seem to be some form of analogical (if not romantic) reasoning. ACM's (2012) centenary celebration of Turing by all the living Turing-award winners makes the point quite clear: "This development [algorithmic thinking outside computer science] is an exquisite unintended consequence of the fact that there is *latent* computation underlying each of these phenomena [cells, brains, market, universe, etc.], or the ways in which science studies them." [emphasis added]

Nature-inspired computing is not to be confused with 'nature computes' movement. Personally, I would not lose too much sleep if some problems are not amenable to computationalist understanding. Discovery by method has its limits.

select the strands that stick out. We can do this by taking out the longest one and repeating the procedure of measurement without having to bang the sheaf on the table again. We sort by length in the end. This method is linear on the number of n rods, which is better than the algorithmic complexity of $O(n \log n)$ at best.[7]

Both methods are based on a comparison of values. However, the "analog variant" will not work with liquids or gas. What we show by this method is an understanding of gravity and solidity, not sorting. Although it appears to be algorithmic, it does not lead to a computational understanding of the problem but to a particular solution. If we want to address the sorting problem in an analog way, we must provide analogs which fix the structure of the problem via equations.

Attempting a computational explanation reveals that *comparison sort*, which is a family of algorithms that establish an order in the data by comparison of values to be sorted, and other types, such as distribution and radix sort, have different implications in exploiting the domain properties, for domains which are amenable to be ordered in various ways (Knuth 1973). What we understand is the sorting domains. In fact, we create abstract and physical domains with that understanding, such as the zip code; see Knuth (1973):177.

Narrow interpretations of the finiteness of an algorithmic process must also be analyzed in this context, as a question of whether we try to understand the problem captured by the algorithm.

Burgin (2001) suggests that operating systems (OS) are algorithms, and for them to do their work they should never stop. "The real result of an OS is obtained when the computer does not stop (at least, potentially). Thus, we conclude it is not necessary for an algorithm to halt to produce a result."*ibid*:86. An OS does not have an *end* result; therefore, it is not expected to deliver a completion for itself, but surely, its intermediate results, namely execution of other programs, must be finite, because someone needs their results. An OS is successful as long as it delivers finitely computable results when the processes it runs are finitely computable. The explanation, again, is the computation cycle. We allow the abstract subprocesses of an OS to continue their evolution, which is their execution; then, take their results and check for the success of the OS for delivering a result, which is different from the correctness of the subprocesses. As an OS continues to evolve nobody asks it the question whether it has delivered results for all its possible input processes. That question still has no algorithmic solution, but we can live with that because we did not ask the designer that question.

2.3 Understanding the Computational Nature of a Problem

In the light of what has been covered so far, Searle (1990) presents one interpretation of computing which is untenable. According to Searle, a wall can execute the

[7] Algorithmic complexity theory is based on Turing's notion of 'next', such as the next step, next state, and next input. Problem size in the theory is measured with this concept. It is not a physical concept, but obviously physically realizable; see Bozşahin (2016) for more philosophical implications.

program *wordstar* if formal symbol manipulation suffices to compute, because the wall is complex enough to embody the formal structure of *wordstar*.

Of course, it can. For it to do that, it has to contain primitives that can execute the program, *and* a mechanism to turn the formal structure of the program to executable instruction, in which case it would be a brick-implemented computer, rather than a brick wall. If it does not have them, project *wordstar* onto the wall until eternity and you will never get a response. Somebody has to have a theory about 'computing walls' before we start expecting something from them by subjecting them to data.

There are also digital computers which translate physical properties to formal structure in different ways (by the process which we called *decoding* in footnote 4). Some do their work by translating light intensity to abstract entities such as 1s and 0s, rather than voltage. This is called *optical computing*. There are also computers which manipulate the water flow in a pipe for abstraction, which are more experimental. They map these information levels to 1s and 0s; so, we cannot equate digital computing with electronic computing.

Pask brought a whole new (and currently neglected) dimension to analog experimental devices, with his Colloquy of Mobiles (Pask 1968). It is literally a purely mechanical device. The mobiles communicate with each other using light and sound, and with people, who use flashlights and mirrors. They can interact with bodies to do things without abstraction, giving rise to male and female as behaviors. Although Pask was careful not to call them computers, they arise from the intensional view in the sense promoted here because they are simple but not simplistic mechanical attempts at an explanation using time course evolution of responding bodies (see Cariani (1998) for its realization using analog representations). The idea behind Pask's machines is that because they have underspecified goals, they can produce ecological novelty if the machine and the human choose to actively participate in completing the deliberatively unspecified values for the degrees of freedom. These degrees of freedom, however, are carefully designed into the mechanism, and they depend on a conversational maxim, which is an equation.

The reverse trend, de-computationalizing a computational problem, does not help us understand the problem, because it is unacceptably unclear about the mechanism. The perennial $P = NP$? question has been given a purported proof in the affirmative by Bringsjord and Taylor (2005), which the first author himself finds puzzling because he believed $P \neq NP$.[8] They take an NP-complete problem, the Steiner Tree,[9] and show an analog realization (it is not clear whether this includes every instance of it), which is then solved by analog techniques linearly, using soapfilm processing.

[8] Problems in P have polynomial solutions, where polynomial is on the problem size in the sense of footnote 7. NP problems can check a given solution with polynomial effort. Whether P and NP are the same is an open problem in computer science, with serious implications in economics, social sciences and natural sciences; see Fortnow (2013) for an entertaining coverage of these aspects.

[9] To avoid a cryptic mathematical exposition of the problem, I follow the authors' informal description: The Steiner Tree Problem is equivalent in real life to building a road system of minimum length for n towns, possibly making intersections outside of towns. If the number of vertices that can be formed outside of towns (called Steiner vertices) is not known, the problem is NP-hard. If the question is whether we have a solution with length m then it is NP-complete. We are discussing the latter problem.

In doing so the authors introduce a physical possibility operator into the expression language, namely logic, which is also the inspiration for the operator, from the modal logic's possibility operator. The proof has modalized and unmodalized versions.

Let us take their word for it that the logical part of the argument is correct. The physical part of the experiment was put to test beyond the idealized world employed by 'nature computes' enthusiasts. Aaronson (2005) built its physical world with soap, pins, and two glass plates to hold the pins. He reported that the solutions stopped coming for soapfilm processing of the Steiner Tree problem after a certain small size. Maybe the problem *was* solved but we could not arrive at a point where we could check all of them ('Check' is the keyword here because the class *NP* is defined with this term.) Having access to a solution is a necessary stage for a computational understanding of the problem because, whether solved digital or analog, we have to decode (translate) the answer back to the questioner. Even eternally running OSs do that for their client programs, including real-time OSs running chemical plants based on analog decisions.

Worse still, the soapfilm in Aaronson's experiment did not always provide a solution to the problem for small sizes, nor did it always generate Steiner trees. Can we then retreat to the idealized world to make the proof stick materially? Notwithstanding the fact that this move would be quite counterintuitive to 'nature computes' movement, it has other implications.

The recourse was anticipated by Bringsjord and Taylor, who considered soapfilm exponential speedup to be possible without infinite hardware, because of the assumed ideal analog computing. Since we know that this will lead to infinite energy as we get infinitesimally close to the speed of light, we now have to worry about a physical dilemma. In contrast, keeping the problem computational entails that we conditionalize every proof of problem complexity with 'if $P \neq NP$, then...'. It is a succinct way of stating the current understanding that the world does not consist solely of problems whose solutions can be found just as easily as a given solution can be checked.

Our computational understanding of problems can still improve if we can argue for $P = NP$ in a computational way. Knuth expressed belief in the ACM Turing Centennial in 2012 that it may be true that $P = NP$, and that even so, we will not achieve a constructive proof. This means that even if we realize that finding a solution can be as simple as checking a solution for all *NP* problems,[10] we will not obtain an all-purpose algorithm for doing so; hence, the quest for better understanding of nature and computing will continue. This can be seen in the centennial in Knuth's distinction (reported in Knuth 2014) between known and knowable polynomial-time algorithms versus arbitrary polynomial-time algorithms. Aaronson (2013) elaborates more on philosophical implications.

[10] For example, winning at chess every time in maximum of *n* moves from any starting move would be as easy as testing a checkmate on the board. If the opponent also knows the algorithm, then winning the game reduces to who starts the game. Even a non-player can undertake the test (checkmate condition) without understanding the game, if the rules are given on a piece of paper.

2.4 Part I: Summary

Equating analog computers with nature is a myth. Equating digital computers with computing or digital computers with electronic computers does not help us understand computing itself, or computer's scientific value over and above an instrument. My experience has been that much disparaging, fear *and* glorification of computers arise from reluctance or refusal to see their true scientific value.

When we look at the computer from the perspective of its functionality, we can start with the common understanding that a calculator is a computer which is designed not to learn. The difference between a calculator and a computer is intentional. It is not intensional, because a calculator is a computer restricted only in function, and the properties of a true computer are clear in that it must be *capable* of realizing the universal Turing machine.

We want a calculator to undertake some tasks and not to do so in an unclear or indeterminate functionality. This is true of a pocket arithmetic calculator, a differential amplifier, as well as the computer that sends satellites to orbit Jupiter, modulo errors of measurement and instrument precision. The engineering of these tasks may be quite formidable, and a calculator may be (re)programmable, but there is no unspecified functionality in these tasks.

Today's computers can do more interesting things, but anyone who uses them gains an impression that they are not *designed to learn*. They are designed to execute any computable task, including learning, and these tasks seem to be becoming ever more complex. In the remainder of the summary, I use the intensional view to clarify the distinction between a learning computer and a computer which is built to learn.

The ability to learn, which can range from learning new data to learning to do things the computers were not programmed for, is not the definitive characteristic of today's computers. The physical realizability of computing, however, is a necessary property, even for an abstract computer. Cockshott et al. (2012) showed that neither in Newtonian universe without a finite speed for light nor in relativistic universe, super-Turing computation seems physically possible. Here I add that although physical realizability is not a sufficient criterion to understand a problem computationally, it must be the place to start looking for explanations because many unrealistic alternatives can be eliminated by looking at physical realizability.

If we take computer science as an empirical inquiry, as Simon (1969); Newell and Simon (1976) suggested, then it makes empirical statements to check substantive claims about nature. For a man-made device, this means that although it starts with something as contingent as building on a human convention, it gives rise to behavior which is not so contingent but systematic, reliably observable, and replicable. If this is true, then both its mathematical elegance and physical realizability appear secondary to its scientific role because we can ask the following question: if calculators are subsets of computers which are not allowed to learn, forgive the anthropomorphizing, how does the learning computer learn?

Let us briefly look at game learning in answering this question. Quite recently, a computer program for Go called AlphaGo beat one of the highest-ranked human Go players in the world. Go is a more serious challenge to computers than chess because its branching factor is much larger than chess; therefore, any brute force search method was considered bound to fail in defeating good Go players. Thus, a theory was required; however, it took much less time for AlphaGo to reach to this level than all the previous chess programs, which have been around since the beginning of AI in the 1950s.

What really happened is that, with the amount of computing power we have now, AlphaGo adopted the strategy of starting with human-played games, and then, using the ease of modeling in deep learning systems, it generated massive amounts of games from these base cases as surrogate data and trained itself on them. How do we know about this without having any connection to its designer? The code is not public but the algorithm is; so, anyone can check the mechanism to see whether it stays simple while the learned system becomes more complex.

The process could have started without human-played games, as the TD-Gammon system did for backgammon when it reached the masters level. Can we call these systems learning computers? We can call them computers 'built to learn' if we design these computers with the learning algorithms built-in to their hardware as primitives. The same applies to Computational Intelligence and Evolutionary Computing, in which a probabilistic component and a learning component are indispensable. If these are built-in, then their primitives would be physical instructions implementing what they do about learning and uncertainty. It would mean that now the universal computer has to be formulated as an empirical theory on top of them, as something these primitives can implement. For computational intelligence systems, this means that fuzzy logic, which is usually considered necessary for their functioning and does not have its own learning abilities but can support a system which does, is either built-in or has to be implemented as an empirical theory using the learning primitives. If we implement computational intelligence, evolutionary computing or fuzzy logic in a universal computer, digital or analog, then their learning algorithm is the empirical theory of their content and form, not the primitives of the universal machine. The primitives just assure the theorist that this is doable.

3 Part II: Computers and Content

As mentioned earlier, computer scientists commonly define their field as the practice which tries to answer the following question: "what can be automated?" (e.g. Knuth 1996:3). Due to the Turing machine we can closely connect this *what* question with *how*; thus, a further question concerns to what extent this can be done for an observable domain. This raises another important question as to what computers can say about the kind of content at least some of which is learned, because it has become clear that no purely formal system can answer the basic question about semantic content without infinite regress,[11] and no computer can work as a physically realized

[11] For semantic content, we can take the perspective of the Language of Thought (LOT) hypothesis of Fodor (1975), which requires primitives (or core concepts), to support LOT as its primitives, or its rival, map theory, which states that we have a system of belief maps by which we steer our cognitive functions, rather than individuated sentential statements of LOT. Something has to support the system of belief and the maps. "The brain can do this" is no more an empirical theory than Turing's "being suitably

formal system which makes no use of some non-formal content. In the intensional view, this is an empirical question, rather than a philosophical one. Three examples of this nature are examined below.

3.1 Block and Process Ontology

Block (1978) argues that in its attempt to distance itself from behaviorism, mind functionalism begins to suffer from physicalist chauvinism, which it initially tried to avoid as a program, by excluding some possible physical systems from having mental states. We are told that the reason for this is that "functionalism can be said to 'tack down' mental states only at the periphery–i.e., through physical, *or at least non-mental*, specification of inputs and outputs"*ibid*:264 [emphasis added].

Block is responding to one functionalist statement in which 'behavior' means 'physical behavior'. This attempt is considered to allow very liberal interpretations of mental states *inside* the mind, similar to behaviorists. Machine functionalists' attempt to avoid this by considering Turing-machine states as mental states connecting physical characterizations of inputs and outputs, is said to exclude some systems from bearing mental states; therefore, they are chauvinistic according to Block.

The argument very much depends on machine functionalism's presupposed understanding of mental state. We are told that by such functionalism "each system having mental states is described by at least one Turing-machine table of a specifiable sort and that each type of mental state of the system is identical to one of the machine-table states."*ibid*:267.

This is a category mistake, if not loose talk. A mental state will correspond to a Turing-machine-in-action, if mind-as-TM idea is followed. This is because mental states would be properties of *processes* in the mind-as-TM idea. Turing machines advance the state of computation by action, not just by reference to an ontology. The action must *use* a primitive of the machine to go from configuration to another, not from one TM state to another. Most advocates of machine functionalism would probably subscribe to this view.

To observe the difference between states and configurations in a clearly defined computational process, it is necessary to consider once more the complete definition of a Turing machine given in the beginning of Sect. 2, rather than only looking at its formal component. Here, I repeat the definition: $M = (Q, \Sigma, \Gamma, \Delta, s, h)$, in which Q is the finite set of states that Block refers to. In computer science, the process for M is called *one-step computation*; see Lewis and Papadimitriou (1998) for formal details. It takes a state from Q and a configuration, which is the location of the tape head and what is left on its tape as a combination of $\Sigma \cup \Gamma$ (and some special symbols, such as blank and start point, and let us call the union V for vocabulary). A configuration is usually written as an ordered pair, $(p, u\underline{a}v)$ for some a in V, meaning the tape head

Footnote 11 (continued)
programmed" was for intelligence and understanding. A good place to start for both is by giving them a process ontology. Although map-theorists shun the computer analogy, the intensional view is not an analogy but a constitutive principle, so it might also help those theorists.

is on a, M is in state p, and the tape contains $uav \in V^+$ only. One-step computation is a relation between two configurations established by a single action in Δ. For example, we can go from configuration $(p, u\underline{a}v)$ to $(q, ua\underline{b}v')$ iff Δ specifies that we can go from state p to q when there is *currently* an a on tape, and move to the right (i.e. $v = bv'$ for some b in V). We denote this as $(p, u\underline{a}bv') \vdash_\Delta (q, ua\underline{b}v')$.

This is how a machine with finitely many states can generate countably infinitely many configurations. Obviously, as finite species, we cannot completely recall all these configurations, any more than we can remember all the sentences we have uttered so far and we will have uttered in our lifetime. Note that if we follow the idea of equating mental states with Q above as Block did; i.e., with TM's states, it should be much easier for us to recall more than we think possible.

If machine functionalism is correct, then each of the configurations will be a mental state because of the use of '$\vdash$', assuming that a claim is made that M is a cognitively relevant machine (for vision, planning, language, etc.). In other words, the mental states of M are the configurations engendered by the computations of M, rather than the states of M such as Q.

Clarifying the terms of the computer in this debate shows that *any* computational process has to use '$\vdash$' anyway; so, the real prerogative of machine functionalism lies in empirical claims about M, because only some M's states will always be mental states. This is not chauvinism neither is it an attempt to avoid liberalism of behaviorism by allowing too many Ms to have mental states. The empirical theory must show which Ms have mental content, independent of what substance realizes M. It would lead to a better understanding of M-problems if the theory captured in M can tell us what kind of content can be captured by it and to what extent.

It also follows from this style of thinking that if functionalism is given an operational definition, as has sometimes been undertaken, e.g., "what functions as a mind counts as a mind," then we are back to square one of infinite regress in a different way. To know what *functions* as a mind, we must have an empirical theory about M-problems, and for that we need physically realizable computation. I am not rushing to the defense of machine functionalism or trying to refute it. I only suggest that this is not a convincing way to argue for its weaknesses, as shown by revealed cross purposes when we discuss content, physical states and mental states.

Another argument about content and computers was presented by Searle. From the perspective of the intensional view, it is a weak argument concerning computation.

3.2 Searle's Chinese Room and the Computer

Searle (1980) provided a reenactment of the Turing machine where the logic control unit (Δ in the definition of a TM) is a man. The argument is well-known so I will summarize it briefly; see Searle (2001) for a more detailed statement.[12]

[12] See also Searle's objection to the original replies in the (1980) article, and the volume of Preston and Bishop (2002), containing more new responses. Although Searle also contributed to the volume, he was not given the opportunity to respond to the articles (Preston 2002:46).

We are asked to imagine a room which contains a human being who can only understand English, and the room has one channel of input for Chinese formal symbols and one channel of output for the person inside the room to communicate with the external world which includes native speakers of Chinese. The English speaker has access only to a book of Chinese symbols and their formal correspondences without semantic content in the book.

Searle's argument suggests that if the only input to the room is formal symbols and what happens in the room or what the person inside the room does is only to manipulate the symbols formally; i.e., without bringing *its/his* content into the picture, and in so doing convince a native Chinese speaker that the output is indistinguishable from that of a native speaker by passing the Turing test for being Chinese, then it proves that syntactical computation by itself cannot give rise to understanding of Chinese, because in passing the test the room or the person inside the room has learnt nothing about Chinese. It just handed down one formal symbol in response to another formal symbol.

First I want to draw attention to that this is perhaps the most (only?) productive use of the Turing Test. It clarifies what we are dealing with: The room can be put to test countably infinite times, for Chinese is a countably infinite language, and although this is an impossible physical condition for the thought experiment because the set is endless, we make a negative assertion that something is not possible even if the experiment was successful. This is clearly feasible; however, what is not possible according to Searle is understanding *as such* if we accept the terms of the experiment, because the person inside the room is capable of understanding English, but not Chinese.

What else follows if we accept the terms of the Turing-test part of the experiment? Since we are asked to assume the case in which the person inside the room does not develop an understanding of Chinese, because he does not know the language, and only formally manipulates the symbol correspondents which he can find in the rule book for output (he just needs English, which he knows, to be able to read the book), we must also assume that the rule book as a flat table of correspondences of formal symbols would be infinite. An earlier argument made by Rogers (1959) about the similar experimental conditions did mention that the instructions in the book for such translation must be of finite length but assumed that the paper supply for input-output is inexhaustible, and the room can grow arbitrarily large.[13] The last assumption needs clarification, as given below.

'Infinite' is far different from 'arbitrarily large'. Since we cannot have an infinite-size computing-room without violating the known laws of physics,[14] the infinity that

[13] Rogers (1959):115 provided the first Chinese Room-like argument where the input-output materials are not Chinese symbols but numbers. He was interested in computing number-theoretic functions by a human. He placed a man in the room equipped with a finite set of instructions to compute numbers, who outputted them for checking. He did not suggest a test to decide between man and machine. His test was to be able to compute any number-theoretic function in a finite amount of time. He assumed that the person inside the room is inexhaustible.

[14] *Pace* Copeland (2002), who considers super-Turing computing to be possible, we cannot compute and violate the laws of physics at the same time; see Cockshott et al. (2012) for discussion

we require must be captured by levels of finite abstractions in the rule book, with most likely recursion in them, much like grammar.

This is how compilers translate countably infinitely many arbitrarily large programs of a programming language to executable code; i.e., to *their* semantics. It works because some ground cases of grammar are interpreted without further translation, using the primitive instruction set, which avoids infinite regress.

Whether we are dealing with a wet set of primitives (supporting LOT or belief maps) or a silicon instruction set, or with a sheet of conductive plastic foam as in untranslated analog primitives of Mills (2008), we are not at the syntactic level when we reach them, and no further syntactic translation is possible. These are semantic objects that really "do" things when realized in an architecture with circuits, conductance, light emission, liquid flow, "brain power", etc.

What is syntactic is the reference from the terms of the mechanism to the other terms of the mechanism, and from some terms of the mechanism to some terms of the architecture. This is also true of the extended analog computer, because although equations that show the computational understanding of the problem can be recurrent, they all have to translate, or in technical terms, *to be implicitly contained* (Fig. 1), in the terms of the architecture.

Outside the Chinese room, these primitives cannot be input to the room because we are only allowed to pass formal symbols into that space. Inside the room, we are not allowed to generate these primitives for the person in the room, because the person inside the room uses his knowledge of English *only* to refer to the rule book, which is in English. The Chinese symbols are not interpreted by him. (It could have been language X for him without knowing what X is, since the book itself is in English.) Therefore, in a room like this, and assuming that it passed the Turing test, there must be countably infinitely many realizable spaces, not just an arbitrarily large space, for there is no end to syntactic translation. As far as we currently know this is not physically possible.

Recalling that Turing's infinite tape is a convenient generalization designed to support arbitrarily large amount of space, not infinite amount of space, and any finite-step computation uses finite amount of space in the tape, the problem can be seen to be as follows from the intensional perspective: If finite space-use is not a decidable problem, then finite step-use is not a decidable problem, because we can continue to undertake the translation in that space endlessly. Finite space-use is not a decidable problem, as a corollary of a theorem about Turing machines (see Lewis and Papadimitriou 1998, chapter 5). Therefore, finite step-use is not a decidable problem either. It means that if finite step-use is a decidable problem, then finite space-use is a decidable problem too. Since the person in the room is assumed to be able to respond to any Chinese input, the room and the person are making finite use of steps. It follows that they use finite amount of space. This goes back to the case discussed above toward the end of Sect. 3.1 concerning how a system with finitely many states can engender countably infinitely many configurations. In a finite species, the number of configurations will be arbitrarily large but not infinite.

We cannot offload the formal symbol correspondence problem to countably infinitely many configurations of TM to avoid the infinite-size room problem and expect to solve the empirical question of understanding computationally. TM configurations arise from the use of '$\vdash$', therefore if this use is undertaken by the person inside the room he has mental states to understand the content of the symbol, a case which we are told to avoid. If it is the Chinese room's use of '$\vdash$' for purely formal processing and never by the person inside, we can avoid the infinity problem because we are asked to assume that the room passes the Turing test; therefore, it makes an arbitrarily large but finite use of space in '$\vdash$'. However, now we face the infinite regress problem because we have to show a computer for the room to do '$\vdash$', and if it is a computer then it has some primitives to handle the formal symbols, and these primitives themselves are not syntactic. We seem to be too close to having a grip on the understanding problem to give up on physical computing at this point.

We can return to entertaining the possibility of an arbitrarily large but finite-sized grammar in the room, rather than an infinite space. How can such grammars begin to generate predicates, for we know that understanding a language means understanding the predicates of that language? Predicates have semantic content; therefore, we are looking at the extra-linguistic environment; for example, the environment of the child in the acquisition of grammar. Semantic content was not allowed in the Chinese Room setup as was implicated by the original Robot Reply to Searle;[15] so, how can we expect the room to give rise to grammar in the form of form-meaning correspondences for words and phrases? We just have to prove to Searle that this additional channel is not syntactic in the sense of not being purely formal. If it were syntactic, it would just add more to something that is already assumed to be formal, therefore not very interesting. And to Searle's potential response of that's what he's been arguing for, we can say that we have made the problem a lot clearer than he formulated, by suggesting via intensional explanation that what goes in as hypothesized content into the language acquierer must be interpretable inside, and this is the part that would be unexpected by Searle since it comes from someone who agrees with him: it is true of the child acquiring a language, as well as the computational system that models the process. The difference of the intensional view from the Robot Reply is that a computer with a body would not amount to an empirical theory of language acquisition.

The peculiarity of not having a set of untranslated terms in learning the terms of a language occurred to researchers long before computationalism of this kind. I quote from Fodor (1975):84 for the source:

> "I know of only one place in the psychological literature where this issue [of infinite regress] has been raised. Bryant (1974) remarks: "the main trouble with the hypothesis that children begin to take in and use relations to help them solve problems because they learn the appropriate comparative terms like 'larger' is that it leaves unanswered the very awkward question of how they learned the meaning of these words in the first place" (p.27). This argument generalizes, with a vengeance, to *any* proposal that the learning of a word is essential to mediate the learning of the concept that the word expresses."

15 The Robot Reply suggested that a computer with a body would be like a child learning language.

Some of the concepts, then, are not learned. We do not have to be specific about what they might be to make the argument more convincing. We just have to assume that they must be there to avoid infinite regress of translating syntactic operations to lower and lower terms, because all computations work that way, and they must be rich enough to learn the meaning of any predicate of the language. This is the empirical challenge, and Fodor fought hard for it.

The same is true for the computer, and for Helen Keller, for whom syntax-all-the-way-down arguments have been made for her acquisition of content.

3.3 Keller and Content

One way to comprehend how Keller began to take in content along with the form to give rise to a computable grammar requires some background detail. Rather than rely on external observations, I will present a summary from the personal recollections of Keller (1905) (hereafter *HK*).

Keller lost her sight and hearing irretrievably at the age of 17 months, when she had already spoken her first words (*soup* and *water*). By her own admission, she was on her way to becoming a bitter child because of frustrating failures in being able to express herself, when just before her 7th birthday, a carer, Anne Sullivan, arrived. Sullivan eschewed earlier methods and taught Keller finger spelling. At the end of first day Keller could finger-spell two words, *doll* and *water*; however, she recalls clearly (*HK*:46) that she had difficulty disambiguating *mug* and *water*, although Sullivan trickled water through the palm of her hand as she finger-spelled *water* in the other. On the same page Keller reported an even more striking recollection: when a new doll was placed in one hand when Sullivan finger-spelled *doll* on her other hand, Keller was confused because it was not her usual doll. Keller reported this incident in *HK* as an a-ha moment, when she thinks she realized everything has a name, and that everything became easier after that. I assume this is one of the reasons why Rapaport (2006) considered her to have escaped from the Chinese room; see his original argument, replies, and his reply to replies in (Rapaport 2006; Ford 2011; Rapaport 2011).

We know that Keller was once *not* in a Chinese room, because some hypothesized content had gotten in through language before she became deaf and blind. Afterwards, she is assumed to be in a Chinese Room because of severed take-in of content. Rapaport argues that she went into a Chinese room, and then left it using "syntactic semantics", which is the idea that syntax suffices all the way down to cause semantics in a computer (Rapaport 1988).

Rapaport (1988):84 does discuss a causal link that is necessary in addition to syntax, to give rise to semantics. This link is assumed to be a type of non-syntactic semantics that links the agent to the *external* world by being in it, but according to him it is "not the kind of semantics that is of computational interest." So, clearly, it is not intended to be an inner causal link like an executable primitive. Since the executables are internal mechanisms; in other words, since their existence comes naturally when we adopt a computationalist explanation, we need to question why

Keller cannot do what she was able to do as a correspondence problem, much like the compiler and the problem of translation I exemplified earlier in Sect. 3.2.

There is another way along these lines to understand Keller's joy and a-ha moment while making a less radical jump concerning the content. It is an argument to the best explanation, much like Rapaport's, because we have no way to study Keller. She could have moved from the assumption that *doll* was indexical; i.e., fixed with the referent, to one where it is referential; i.e., indirectly associated with the referent. She already has the knowledge that things *can* be symbolic because she had spoken some of it, before the illness that made her deaf and blind.

This would not be much different in kind than her ambiguity in the first day of Sullivan's experiments about the mug and water. Then, Keller perhaps entertained the possibility that many of her earlier indexical assumptions could also be referential. This would also have been an a-ha experience. The jump here is not so radical if this had been achieved before, and she had been deprived of it later. Recall Keller's surprise when she found that the doll in one hand was not her usual doll when Sullivan finger-spelled 'doll' in her other hand. Instead of $doll'_1$ for the meaning of the word *doll*, where $doll'_1$ is the *index* of her old doll, she would have revised it to $doll'$, or maybe in clearer steps, from $doll'_1$ to $f(doll'_1)$.

The more interesting question is raised by the intensional view: How did Keller know that what Sullivan does to one hand when finger-spelling what happened in the other one is semantics, but not syntax? If she really were in a Chinese room, she could not have known that. This is because the conditions of the Chinese room experiment is such that only formal symbols are allowed to enter the room. If Rapaport's argument is that she was once in the Chinese Room, then Keller could only take formal symbols inside, even if we follow his assumption that being in a physical world is a causal link. If this semantics is "of no computational interest" then something else has to make the computational system inside physically work.

If we follow Rapaport's escape explanation, then Keller would have to invent generating meanings internally. She can do that if she already knows where to start. Otherwise, we are forced to assume that all children learn the *ability* to associate meanings with form, rather than learn which forms go with which meanings. (Imagine coming to life at the moment when Sullivan came in to Keller's life.) We know since Lenneberg (1967) that the timeline of language development does not differ much across cultures, habits, welfare of the caretakers or the environment, as long as the child can be exposed to linguistic data early on. We would have seen much greater variance if such "learning" takes place for every child.

I submit that Chinese Room's conditions are very unlikely to be part of the truth about relating content with form. It is physically unrealizable for adults (and for "rooms") who are assumed to have passed the Turing Test or it is unlikely because anyone who has an inborn capacity to indirectly associate forms with meanings will have a grip on the correspondence problem by physical computing. The latter case never takes in only the form anyway to achieve understanding; a hypothesized meaning is also taken in. What avoids infinite regress here is the intensional consideration that it can not be turtles all the way down;[16] so, those meanings will be interpreted by the virtue of syntax taking them to ground cases where no further syntactic translation takes place.[17]

3.4 Part II: Summary

The argument I presented concerning the Chinese Room may appear to vindicate Searle, but actually it does not. What some of us have always thought about Searle, more worrying than his reasonable claim that humans are humans, computers are computers, both can be very intelligent, but only the former has causal powers of the brain, has been revealed as apathy in the following more recent statement: "The fundamental claim [of the Chinese Room argument] is that the purely formal or abstract or syntactical processes of the implemented computer program could not by themselves be sufficient to guarantee the presence of mental content or semantic content of the sort that is essential to human cognition." (Searle 2002:51). This is true but rather pessimistic about the explanatory role of computers, and it seems to suggest that they should know their place as instruments. Searle does not help us understand cognition: "In this science [of consciousness and intentionality] the computer will play the same role that it plays in any other science. It is a useful tool for investigati on"*ibid*:68.[18]

If we can suggest a correspondence problem which is physically realizable about associating linguistic form and meaning, then *any* system capable of physically realizable computing can be cast to face the same problem as the child, provided we find the right theory and model for the two ends of the correspondence. Even that is not sufficient for the computer to have a conscious experience. Its job is to explain the process. It seems easier to just drop the "(if) computers (still) can (not) do this" kind of rhetoric. The mechanism suggested would be sufficiently clear if we adopted the intensional view. Anyone can peek inside to see if the algorithm is cheating. (One transparent algorithm is given in Abend et al. 2017.) If it is not possible for the machine to confront these circumstances, then it is not possible to computationally

[16] I first saw the use of this mythological 'word turtle' idea in the sense closest to the current discussion in Ross (1967), who attributes it to William James. Its significance for the intensional view is that Ross recalls it with a "bull's eye relevance to the study of syntax."

[17] Without this assumption, we would not be too far from linguistic relativism of the Sapir variety, in which we would think in a language. Although there are many cases in which linguistic terminology is deeply cultural; for example, basicness of color terms, this is not a causal link from language to thought, because we have seen tragic cases of having thought but not language, at least quite unexpected cases of inferential ability (if language caused thought), in the light of little (almost no) linguistic exposure up to puberty. One such case is Genie (Fromkin et al. 1974; Curtiss et al. 1974). Assuming that language is an expression of thought seems to avoid these problems. In the current argument, it is not only empirically on better grounds, but also a necessary consequence of thinking that language is a computational mechanism. On the other hand, syntactic semantics require it as an extra assumption; but, let me stress that nobody is denying the role of compositional semantics in syntax. The question is whether syntax alone can cause semantics.

[18] Ford (2011):70, who defended Searle's views against Rapaport, is less apathetic, but still analogical in thinking about computers: "if we can get a computer to have meaningful conscious experiences—the road to natural language acquisition and understanding would be clear (as far as Searle is concerned)."

study the acquisition environment of the child either, in linguistics or in psychology. My guess is that Searle would not be prepared to take that step.

Whether it tackles a simple or complex problem, the mechanism of the computer can be kept simple, and that can be observed all the way. It does not follow that any algorithm we chose to implement, or its physical realization in the form of an architecture if the algorithm is well-understood is simplistic. These models must prove their explanatory worth. The simplicity of the mechanism is not a guarantee for the explanatory value of the model. What it does is to keep the terms of the theory- and model-building clear so that we can attempt an explanation.

I believe the process of theory and model building within the intensional view proves its worth in assessing Block's argument and Keller's case. Keller understood what it means to have words with meanings, and she did so presumably by trans-calculatory computation, by bootstrapping her system from Anne Sullivan's *two* pieces of information, one on each palm, both of which must be interpretable inside. The story of the syntactic piece of information is well-known from analogies to purportedly pure formal processing by a Turing machine. The semantic piece of information could not have worked by itself, using content all the way, because if it worked it would mean that the blind and deaf Keller could have done this before her carer arrived when she was exposed to wind, grass, water, etc. She did not do this because she needed a medium in which she could solve the correspondence problem herself. Nobody had provided a possible medium for *that* before Sullivan arrived.[19]

4 Conclusion

Digital computers cannot be syntax all the way down because they need untranslated terms to physically compute. Analog computers cannot be content all the way up because their results have to translate to some terms to be of any use. The difference between them is not allowing measurement errors to accumulate in determining the physical state of computation (the digital case), or allowing them to accumulate until the point of translation (the analog case).

Computer science is not about content *per se* or syntax. It is about connecting the two without infinite regress, which is to say in a physically realizable way. The correspondence is not necessarily algorithmic neither is it necessarily analogical. Practice tries to make clear the terms of the mechanism of the correspondence so that empirical theories can be built upon them.

Computers are known to be simple mechanisms, but they are not simplistic. That makes them useful tools, novel discovery mechanisms, and also a political force.

[19] Keller's case is different from that of a child who is deaf *or* blind in the critical period of acquisition. The child in these circumstances can have some access to meanings out there by his own initiative, to relate them to forms. In fact blind children create form differences for the semantic distinction of *look* and *see*, although they cannot experience visual looking or visual seeing. Deaf or hearing children who are born to deaf parents acquire their sign language in the normal time course of language acquisition. Blind children follow a normal course too as long as they are exposed to language; see Gleitman and Elissa (1995) for a summary.

The key to their political force is the disproportionate complexity of what we can do with their simplicity. Nobody is afraid of calculators, simple or complex, although some of them—those that fly airplanes for example—may give us cause for concern. We can imagine what degrees of freedom they are allowed to exploit because of design choices that can be made very explicit. The transparency of the terms and the mechanism will enable us to take informed stances in relation to a computer model if we have access to that transparency. Manipulating the transparency has been a factor in shaping the impact of their political force, as can be observed often these days. I would argue that people, professionals and non-professionals alike, instinctively adopt the intensional view of computers when they use them to rise above data in a correspondence problem.

Acknowledgements I thank five reviewers of Minds and Machines for their comments, which significantly improved the paper. Thanks also to Halit Oğuztüzün and Umut Özge for feedback on an earlier draft, which led to some new sections, and to Vincent Nunney for last minute help with some pieces of text. All errors and misunderstandings are mine.

References

Aaronson, S. (2005). Guest column: NP-complete problems and physical reality. *ACM SIGACT News*, *36*(1), 30–52.

Aaronson, S. (2013). Why philosophers should care about computational complexity. In B. J. Copeland, C. J. Posy, & O. Shagrir (Eds.), *Computability: Turing, Gödel, Church, and beyond*. Cambridge: MIT Press.

Abend, O., Kwiatkowski, T., Smith, N., Goldwater, S., & Steedman, Mark. (2017). Bootstrapping language acquisition. *Cognition*, *164*, 116–143.

ACM. (2012). ACM turing centenary celebration. Association for Computing Machinery, June 15–16, San Francisco. http://turing100.acm.org/.

Bickhard, M. H. (1996). Troubles with computationalism. In W. O'Donohue & R. Kitchener (Eds.), *Philosophy of psychology* (pp. 173–183). London: Sage.

Block, N. (1978). Troubles with functionalism. In C. W. Savage (Ed.), *Minnesota studies in the philosophy of science*. Minneapolis: University of Minnesota Press.

Bozşahin, C. (2016). What is a computational constraint? In V. C. Müller (Ed.), *Computing and philosophy. Synthese Library 375* (pp. 3–16). Heidelberg: Springer.

Bringsjord, S., & Taylor. J. (2005). An argument for $P = NP$. arXiv:cs/0406056.

Bryant, P. E. (1974). *Perception and understanding in young children*. New York: Basic Book.

Burgin, M. (2001). How we know what technology can do. *Communications of the ACM*, *44*(11), 82–88.

Cariani, P. (1998). Epistemic autonomy through adaptive sensing. In *Intelligent control (ISIC). Held jointly with IEEE international symposium on computational intelligence in robotics and automation (CIRA), Intelligent systems and semiotics (ISAS)* (pp. 718–723).

Cockshott, P., Mackenzie, L. M., & Michaelson, G. (2012). *Computation and its limits*. Oxford: Oxford University Press.

Copeland, B. J. (2002). Hypercomputation. *Minds and Machines*, *12*(4), 461–502.

Copeland, B. J., & Shagrir, O. (2011). Do accelerating Turing machines compute the uncomputable? *Minds and Machines*, *21*(2), 221–239.

Curtiss, S., Fromkin, V., Krashen, S., Rigler, D., & Rigler, M. (1974). The linguistic development of Genie. *Language*, *50*(3), 528–554.

Dennett, D. C. (1971). Intentional systems. *The Journal of Philosophy*, *68*(4), 87–106.

Dennett, D. C. (1991). *Consciousness explained*. New York: Little Brown & Co.

Dewdney, A. K. (1984). On the spaghetti computer and other analog gadgets for problem solving. *Scientific American*, *250*(6), 19–26.

Fodor, J. (1975). *The language of thought*. Cambridge, MA: Harvard.

Ford, J. (2011). Helen Keller was never in a Chinese Room. *Minds and Machines*, *21*(1), 57–72.

Fortnow, L. (2013). *The golden ticket: P, NP, and the search for the impossible*. Princeton: Princeton University Press.

Fromkin, V., Krashen, S., Curtiss, S., Rigler, D., & Rigler, M. (1974). The development of language in Genie: A case of language acquisition beyond the "critical period". *Brain and Language*, *1*(1), 81–107.

Gandy, R. (1980). Church's thesis and principles for mechanisms. *Studies in Logic and the Foundations of Mathematics*, *101*, 123–148.

Gleitman, L. R., & Elissa, L. N. (1995). The invention of language by children: Environmental and biological influences on the acquisition of language. In L. R. Gleitman & M. Liberman (Eds.), *Language: An invitation to cognitive science* (2nd ed., pp. 1–24). Cambridge, MA: MIT Press.

Graham, P. (1994). *On Lisp*. Englewood Cliffs, NJ: Prentice Hall.

Horsman, C., Stepney, S., Wagner, R. C., & Kendon, V. (2013). When does a physical system compute? *Proceedings of the Royal Society A*, *470*, 20140182.

Hoyte, D. (2008). Let over lambda. HCSW and Hoytech. Doug Hoyte. ISBN 9781435712751.

Jay, B., & Given-Wilson, T. (2011). A combinatory account of internal structure. *The Journal of Symbolic Logic*, *76*(3), 807–826.

Keller, H. (1905). *The story of my life*. Garden City, NY: Doubleday.

Knuth, D. E. (1973). *Searching and Sorting, the art of computer programming, vol. 3*. Reading, MA: Addison-Wesley.

Knuth, D. E. (1996). *Selected papers on computer science*. Cambridge: Cambridge University Press.

Knuth, D. E. (2014). Twenty questions for Donald Knuth. http://www.informit.com/articles/article.aspx?p=2213858. Accessed 1 June 2017.

Lenneberg, E. H. (1967). *The biological foundations of language*. New York: Wiley.

Lewis, H. R., & Papadimitriou, C. H. (1998). *Elements of the theory of computation* (2nd ed.). New Jersey: Prentice-Hall.

Mills, J. W. (2008). The nature of the extended analog computer. *Physica D: Nonlinear Phenomena*, *237*(9), 1235–1256.

Newell, A., & Simon, H. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, *19*(3), 113–126.

Pask, G. (1968). *Colloquy of mobiles*. London: ICA.

Piccinini, G. (2008). Computers. *Pacific Philosophical Quarterly*, *89*, 32–73.

Pitowsky, I. (1990). The physical Church thesis and physical computational complexity. *Iyyun: The Jerusalem Philosophical Quarterly*, *39*, 81–99.

Preston, J. (2002). Introduction. In Preston and Bishop (2002).

Preston, J., & Bishop, M. (Eds.). (2002). *Views into the Chinese room: New essays on Searle and artificial intelligence*. Oxford: Oxford University Press.

Rapaport, W. J. (1988). Syntactic semantics: Foundations of computational natural-language understanding. In J. H. Fetzer (Ed.), *Aspects of artificial intelligence* (pp. 81–131). Holland: Kluwer.

Rapaport, W. J. (2006). How Helen Keller used syntactic semantics to escape from a Chinese Room. *Minds and Machines*, *16*(4), 381–436.

Rapaport, W. J. (2011). Yes, she was!. *Minds and Machines*, *21*(1), 3–17.

Rogers, H, Jr. (1959). The present theory of Turing machine computability. *Journal of the Society for Industrial and Applied Mathematics*, *7*(1), 114–130.

Ross, J. R. (1967). *Constraints on variables in syntax*. Ph.D. dissertation, MIT. Published as Ross 1986.

Ross, J. R. (1986). *Infinite syntax!*. Norton, NJ: Ablex.

Rubel, L. A. (1993). The extended analog computer. *Advances in Applied Mathematics*, *14*(1), 39–50.

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

Searle, J. R. (1990). Is the brain's mind a digital computer? *Proceedings of American Philosophical Association*, *64*(3), 21–37.

Searle, J. R. (2001). Chinese Room argument. In R. A. Wilson & F. C. Keil (Eds.), *The MIT encyclopedia of the cognitive sciences* (pp. 115–116). Cambridge, MA: MIT Press.

Searle, J. R. (2002). Twenty-one years in the Chinese Room. In Preston and Bishop (2002).

Shagrir, O. (1999). What is computer science about? *The Monist*, *82*(1), 131–149.

Simon, H. (1969). *The sciences of the artificial*. Cambridge: MIT Press.

Turing, A. M. (1936). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, *42*(series 2), 230–265.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.