Computers aren't syntax all the way down or content all the way up

1 Introduction

The first part of the article argues that computers cannot be syntactic machines all the way down. They have to have non-syntactic primitives, both in their abstract form and their physical form, to be able to carry out their syntactic processing. This part also shows that defining computers as calculators, and digital computers and analog computers as different kinds of computation—in fact any different realization of computers as something different in kind, is not very helpful in understanding what we can do with their capacity. They are different ways of implementing one idea of a computer.

The second part argues that the results of the first part may or may not be good news to computationalism as currently conceived, or machine functionalism in general. Searle was right to reject the Turing test as a test for understanding, but in doing so he misrepresented the idea of a computer. And Turing seems to have made matters worse earlier by unfortunate comparisons of 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 mind as a computer. An empirical theory of content has to be provided with the computer, not appealing *only* to the core functions of the computer, but, crucially, reducible to their execution. This I believe is good news because it can lead to a research program. The bad news 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. We suggest that the whole idea of machine functionalism is question-begging. The intensional view of the computer as the basis for theory-making may break the circle. The idea is based on the following observations.

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 ei-

ther simple or complex mechanisms is not very exciting, for we know that most of the interesting behavior worthy of studying is complex. Given the current state of science that studying complex behavior with equally complex mechanisms is not all that exciting either, we can narrow down the role of computer science in philosophy to making the terms of computers 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. We are better off working on *how* things related to content can be studied. It 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" (Piccinini 2008:33), do not help us understand what is it 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 some brands of philosophy such as Shagrir (1999), Copeland (2002), Piccinini (2008). For example it differs from Shagrir's 1999:146 definition of computer science: "CS is, indeed, the science of computers, whereas computers are physical dynamics (either algorithmic or not) that are individuated by the formal content of the representations over which they are defined." According to this 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 that is modeled by them.

Even the Church paradigm, lambda-calculus, is too extensional to serve a mechanical explanation in this sense. Its behavioral equivalence to Turing machines is sometimes not enough. We look for more expressivity. For example, there is a Turing-computable function that distinguishes the two lambda-calculus functions SKK and SKS, but the difference cannot be represented by the SK system which gives us the Turing equivalence of lambda-calculus, because SKK and SKS both yield the behavior of the identity function (Jay and Given-Wilson 2011). (The combinators S and K are respectively the lambda terms $\lambda x\lambda y\lambda z.xz(yz)$ and $\lambda x\lambda y.x$.) Such functions are Turing-computable intensional functions that are not representable by SK.

The implication is that there is more to computing than which can answer the behavioral question, for example inspecting its own structure. That is why we also define computer science commonly as answering the question "what can be auto-

mated?", knowing that the ensuing *how* question is addressable because of the Turing machine, realized in some physical way, digital or analog.

There are correlates of intensionalism versus extensionalism in the programming paradigms too. All programming languages worthy of the name are Turing-complete, but they are not equally expressive. For example, although many languages provide macros, in languages like Common Lisp we can write macros with which we can write programs that write programs, and inspect the result by programs—macros—before the result is run. The elusive intensional function which we described above seems to be just such a macro-behaving function (this was suggested by Marc Hamann in 2010, in a blog discussion of Jay and Given-Wilson 2011 initiated by the authors).

Graham 1994, Hoyte 2008 explain what we gain from this way of programming. Common Lisp macros are *compile-time* functions that use the same instruction set as run-time functions. Infinite expansion of recursive macros to infinite forms is avoided because macro expansion itself is a built-in function. There are lessons to be had here on infinite regress and physics of computing, from computer science to philosophy. Enter Cartesian theater.

The rest of the paper is organized as follows. Part I tries 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 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 we have so far 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 abstract Turing machine. This can be seen clearly in the mathematical definition of a Turing machine (TM). A TM is defined formally as 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. with reference to Q,Σ and Γ only, 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, in 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. And they are 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,[1] and its 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 to Church's lambda calculus. It 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 are 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' here are abstract but physically realizable concepts. In an architecture, they will have a physical correlate, using the representation relation which we describe in the next section, in footnote 2. This mapping is required because the physical evolution of computing (i.e. computer's execution) is done at the physical level, with physical objects; see §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. Bickhard (1996). 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, for what I called a blueprint above. Turing did not switch back and forth between architectures and mechanisms in talking about the Turing machine, except perhaps in his discussion

[1] Except possibly Gödel 1946, if Wang 1974:325 is right. He was always skeptical about taking formal definitions too seriously, yet could not help thinking that Turing's definition captured something profound.

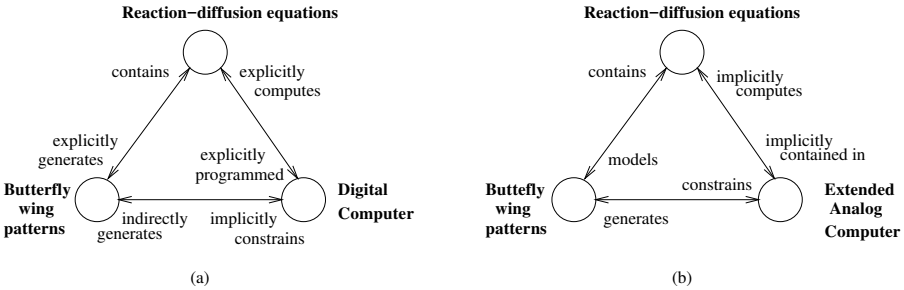

Fig. 1 (a) Digital and (b) analog computer paradigms, respectively giving (a) algorithms and (b) analogs for butterfly wing morphogenesis, from Mills 2008. Objects are in bold, relations are in normal font.

of Oracle machines, which are idealized, and which 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 made references to space, he made no reference to physical time. His space depends on the notion of 'next' only (next tape cell, next input, next state), which is an abstract term referring to a mechanism, not a physical term. Therefore the talk of the purist's view of the Turing machine, and the realist's view of it, 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. That is why many of us think Turing's definition seems to capture the limits of computability convincingly.)

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

However, analog computers are programmable too, such as the extended analog computer of Rubel (1993), but its programming requires a good deal of scientific and engineering expertise at the moment. 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) reports a Pitowsky (1990) experiment, where 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 $(k+m+n)/3$.

Thermodynamics gives us instant "computation" of the average for any number of containers, according to this view.

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. I think a more pressing question is whether it is computation at all. We cannot repeat that experiment with gas, we cannot do it in outer space, nor can we do it with non-conducting material, solid or liquid. What we seem to show an understanding for by this procedure is the nature of thermodynamics, heat conductance, and living on earth, but not through computation.

What we understand is averaging itself when we map these quantities to (say) numbers via a representation relation,[2] 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 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 Figure 2.[3] In cases where it does not predict the expected outcome, we can see the instrumental value of computer science. Maybe (i) our theory of averaging was bad, or (ii) the compiler which translated our algorithm which 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 encoding and decoding which depend on it would also be useless. If the computer is well-designed, and the compiler (equivalently, the implicit relations of the equations and the extended analog computer), then the second to fourth cases would be known not to hold, so that we go back and change our theory of averaging by revising our algorithm or equations.

[2] 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.

[3] 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 Figure 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 Figure 1, as in Figure 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 correct algorithm is much like the diagram in Figure 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.

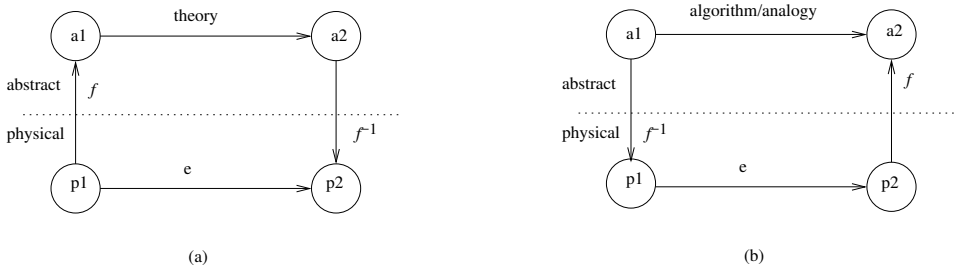

Fig. 2 The modeling relation (b), from computer science, and reversing the modeling relation (a), from physics, adapted from Horsman et al. 2013. 'e' is for evolution/progress of the physical system, 'a' is abstract state, and 'p' is physical state. f is the representation relation. Use of f is decoding,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

It is difficult to see how, without decoding and encoding, that is, without modeling, using nature by itself, we could find out what went wrong. It is possible if we are clear at every level about what pieces can do what, and how. It does not 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:ch4. (It is just that 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 sets the causal relation for it too. 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 do the computation.

Analog computers do not just leave the calculated result as such to finish their work. They need to translate back—called *implicitly compute* in Mills 2008 and in Figure 1—to the terms of the equations, for result to be of any value. 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 Pitowsky (1990) experiment is the translation of the analog results to be of any value. The question is, to what? What are they translated into? Surely an abstract domain, whether it is represented as an algorithm or an equation. Therefore, the choice we are forced to make in Shagrir (1999) about either leaving analog computers aside if we want to talk about compu-

tation today, or assume that everything in nature computes, is a false dichotomy.[4] I think making the terms of computation clear does the 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 be studied without algorithms as well. Dewdney (1984) proposed a method inspired by nature to do it the "analog way". (As we shall see, it may be analog, but it is not clear whether it is a computational analog—an equation for an analog computer.) We can take a sheaf of spaghetti, bang it on a table, and pick the rods that stick out. We can do this by taking the longest one out and repeating the procedure of measurement, without having to bang them on the table again. We sort by length in the end. This method is linear on the number of n rods, which is much better than the algorithmic complexity of $O(n \log n)$ at best.

Both methods are based on comparison of values. However, the "analog variant" is not going to 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 a particular solution. If we want to address the sorting problem in an analog way, we must come up with analogs which fix the structure of the problem via equations.

Attempting a computational explanation reveals that comparison sort and other kinds, for example distribution and radix sort, have quite different implications for exploiting the domain properties, for domains amenable to order in various ways (Knuth, 1973). What we understand is the sorting domains. In fact, we create some 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 be analyzed in this context as well, 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 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 let abstract subprocesses of an OS to 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 than the correctness of the subprocess). And

[4] 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." [my emphasis]
Personally, I would not lose too much sleep if some problems are not amenable to computationalist understanding. Discovery by method has its limits.

we continue to evolve the OS. Nobody asks an OS the question whether it has delivered results for all its possible input processes. That question still has no algorithmic solution, but we can live that with because we did not ask this question to its designer.

2.3 Understanding the computational nature of a problem

Searle (1990) delivers one interpretation of computing which is untenable, in light of what we have covered so far. A wall can execute the program *wordstar*, according to him, if formal symbol manipulation suffices, because it 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 wall. If it does not have them, project *wordstar* onto it 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, by the way, digital computers which translate physical properties to formal structure (by the process which we called *decoding* in footnote 2) in different ways. Some do their work by translating light intensity to abstract entities like 1s and 0s, rather than voltage. It is called *optical computing*. There are also those which manipulate 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.

Gordon Pask brought a whole new (and currently neglected) dimension to experimental devices, with his Colloquy (Pask, 1968). It can interact with bodies to do things without abstraction, giving rise to male and female as behaviors. Although he 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 a realization using analog representations. The point of his machines is that because they have underspecified goals, they can produce ecological novelty if the machine and the human choose to actively participate to fill in 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 one of the authors himself find puzzling because he believed $P \neq NP$. They take an *NP*-complete problem, Steiner Tree,[5] and

[5] 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.

show an analog realization (for every instance of it—we are not sure), which is then solved by analog techniques linearly, using soapfilm processing. In doing so they introduce a physical possibility operator into the expression language (logic, which is also the inspiration for the operator, from modal logic's possibility operator). The proof has modalized and unmodalized versions in logic.

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 which 'nature computes' enthusiasts employ, including Dewdney. Aaronson (2005) builds its physical world with soap, pins, and two glass plates to hold the pins. He reports 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 get to a point where we can check all of them. (And 'check' is the keyword here because the class *NP* is defined with it.)

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 which run chemical plants based on analog decisions.

Worse still, the soapfilm in Aaronson 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. Soapfilm exponential speedup is considered possible by them 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$\cdots$'. 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 has expressed belief in ACM Turing Centennial in 2012 that it may be true that *P=NP*, and that even so, we will not get a constructive proof. It means that even if we realize that finding a solution can be as simple as checking a solution for all *NP* problems,[6] we will not get an all-purpose algorithm for doing so, hence the quest for better understanding of nature and computing will continue. Notice in the centennial 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.

[6] For example, playing chess would be as easy as testing a checkmate on a board, to win all the time in maximum n moves from any starting move. If the opponent also knows the algorithm, then chess reduces to who starts the game. Right now even a non-player can do the latter test (checkmate condition) without having to make an effort to understand 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, do 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 also intensional, because the properties of a true computer are clear: it must be *capable* of realizing the universal Turing machine.

We want a calculator to do some tasks, and do so in no 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 unclear subtask in these tasks.

Today's computers can do more interesting things, but anyone who uses them gets the impression that they are not *designed to learn*. They are designed to execute any computable task, and the tasks seem to be getting ever more complex.

Therefore the ability to learn, which can range from learning new data to learning to do things they weren't programmed for, does not seem to be the definitive characteristic of computers.[7] Physical realizability of computing, however, is a necessary property even for an abstract computer. Cockshott et al. (2012) show that either in Newtonian universe without finite speed for light, or in relativistic universe, super-Turing computation seems physically impossible. I will 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 it.

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 some human convention, it gives rise to behavior which is not so contingent but systematic, reliably observable and replicable.[8] If so, then both its mathematical elegance and physical realizability begin to look secondary compared to its scientific role: If calculators are subsets of computers which are not allowed to learn—pardon the anthropologizing, then how does the learning computer learn?

Let us look at the game of Go in answering this question. Quite recently, a computer program for Go called alphaGo beat one of the highest-ranked human Go player in the world. Go is a more serious challenge to computers than chess because its

[7] We say more about this property in relation to nature-inspired computing, which is not to be confused with 'nature computes' movement, toward the end of this section.

[8] Curry 1951/1970:56 considered mathematics an empirical science for much the same reason, suggesting that its essence does not lie in a particular formal system, "but in formal structure as such."

branching factor is much larger than chess, therefore any brute force search method was considered bound to fail in beating good Go players. A theory was needed. However, it took much less time for alphaGo to come to this level than all the chess programs before, which have been around since the beginning of AI in 1950s.

What really happened is that, with the amount of computing power we have now, alphaGo took 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 that, although we may have no relation to its designer? The algorithm is public, not the code, and anyone can check the mechanism to see whether it stays simple while the learned system becomes more complex.)

We could have started without human-played games, as TD-Gammon system did for backgammon when it reached the masters level. Can we call these systems learning computers? If we design the computer with the learning algorithms built-in to its hardware as primitives, we can. The same applies to computational intelligence and evolutionary computing, in which a probabilistic component and a learning component are indispensable. Then their primitives would be computational statements about what they can do, which means 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, it means that, fuzzy logic, which is usually considered necessary for their functioning, and does not have its own learning abilities but it 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 them 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 define their field commonly as the practice which tries to answer the following question: "what can be automated?" (e.g. Knuth 1996:3). Because of the fact that we can connect this *what* question with *how* very tightly, due to the Turing machine, a natural question to ask is to what extent it can be done for an observable domain. Then, an important question that computers raise is what they 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,[9] and no computer can work as a physically

[9] For semantic content, we can take the perspective of the Language of Thought hypothesis (LOT) of Fodor (1975), which requires primitives (or core concepts), to support LOT as its primitives, or its rival, map theory, which says 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 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 help them too.

realized formal system which makes no use of some non-formal content. In the intensional view, this is an empirical question, rather than philosophical. We look at three examples of this nature.

3.1 Block and process ontology

Block (1978) argues that mind functionalism, in its attempt to distance itself from behaviorism, begins to suffer from physicalist chauvinism, which it tried to avoid as a program, by excluding some possible physical systems from having mental states. The reason for that, we are told, is because "functionalism can be said to 'tack down' mental states only at the periphery–i.e., through physical, *or at least non-mental* [my emphasis], specification of inputs and outputs"*ibid*:264. Block is responding in this quote to one functionalist statement that 'behavior' means 'physical behavior'. This attempt is considered to give very liberal interpretations of mental states *inside* the mind, like behaviorists. And machine functionalists' attemp to avoid that by appealing to 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, and the simplest action is called *one-step-computation*. It is a relation between *configurations* of a TM. It does not correspond to an ontological part, such as one of its finite *states*. 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* one-step computation. Most card-carrying machine functionalists would probably subscribe to this view.

To see the difference between states and configurations in a clearly defined computational process, consider again the complete definition of a Turing machine in the beginning of section 2, rather than looking at its formal component only. I repeat the definition: M=$(Q,\Sigma,\Gamma,\Delta,s,h)$. Q is the finite set of states that Block is referring to. The process for M is called *one-step computation* in computer science; 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, 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 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 it 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. (And 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 go by the idea of equating mental states with Q above, i.e. with TM's states, we should be able to recall all that much easier than we think possible.) If machine functionalism is right, then each of these 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 claim of machine functionalism lies in empirical claims about M, because only some M's states will always be mental states. This is not chauvinism. It is not an attempt to avoid liberalism of behaviorism either, 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 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 sometimes done, 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 am only suggesting that this is not a convincing way to argue for its weaknesses, as shown by revealed cross purposes when we talk about content, physical states and mental states.

Another argument about content and computers came from Searle, and, from the perspective of the intensional view, it seems to be a weak argument.

3.2 Searle's Chinese room and the computer

Searle (1980), and much earlier, Rogers (1959), whose "squiggle box" is almost exactly the same as "Chinese room", as pointed by Rapaport (2006),[10] provided a reenactment of the Turing machine where the logic control unit is a man. The argument is well-known so I will summarize briefly; see Searle (2001) for a more detailed statement.[11]

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

[10] Block 2002:70 also lays claim to the original argument, from Block 1978. The main mechanics of the argument seems to be from Rogers to the detail.

[11] See Searle's objection to original replies, and the volume of Preston and Bishop 2002, where there are more new responses. Although Searle contributes to the volume as well, he was not given the opportunity to respond to the articles (Preston 2002:46).

access only to a book of Chinese symbols and their formal correspondence to English symbols, without semantic content in the book.

The argument suggests that if the only input to the room is formal symbols, and what the room or the person inside the room does is only to manipulate the symbols formally, i.e. without bringing *its* 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 Chinesehood, 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 some other formal symbol.

First I want to note that it 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 infinitely many 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 were successful. This is clearly feasible. 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, of English, but not Chinese. Fair enough.

What else follows if we accept the terms of the Turing-test part of the experiment? Since we are asked to assume the case where the person inside the room does not develop an understanding of Chinese, because he does not know the language, and only formally manipulates the symbols correspondents of 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 correspondence of formal symbols would be infinite. (The original argument by Rogers does mention that the instructions in the book for this translation must be of finite length, but assumes that the paper supply for input-output is inexhaustible, and the room can grow arbitrarily large. The latter assumption needs clarification.)

'Infinite' is a whole lot different than 'arbitrarily large'. Since we cannot have an infinite-size computing-room without violating the known laws of physics (because we know, *pace* Copeland 2002, that we cannot compute and violate the laws of physics at the same time—see Cockshott et al. 2012 for discussion), the infinity that we need must be captured by levels of finite abstractions in the rule book, with, most likely, recursion in them, much like a 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 the 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 untranslated analog primitives of Mills (2008), we are not at the syntactic level when we get to 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 terms of the mechanism, and from some terms of the mechanism to some terms of the architecture. (This is true of the extended analog computer too, because although the equations that show the computational understanding of the problem can be recurrent, they all have to translate—*to be implicitly contained in* is the technical term in Figure 1—to the terms of the architecture.)

Outside the Chinese room, these primitives cannot be input to the room because we are allowed to pass formal symbols only. Inside the room, we are disallowed to generate these primitives for the person in the room because the person inside the room uses his knowledge of English *only* to look up the rule book, which is in English. 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 space, not just arbitrarily large space, for there is no end of syntactic translation. This is not physically possible as far as we know.

Recall that Turing's infinite tape is a convenient generalization which is designed to support arbitrarily large amount of space, not infinite amount of space usage. Any finite-step computation uses finite amount of space in the tape. It can be seen as follows: If finite space-use is not a decidable problem, then finite step-use is not a decidable problem, because we can continue to do 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:ch5. 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 where we discussed toward the end of §3.1 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 the TM to avoid the infinite-size room problem, and expect to solve the empirical question of understanding computationally. TM configurations arise from use of '$\vdash$', therefore if this use is done 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 arbitrarily large but finite use of space in '$\vdash$'. But 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.

We can go back to entertaining the possibility of an arbitrarily large but finite-sized grammar in the room, rather than 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 acquisition of grammar. Semantic content was not allowed in the Chinese Room setup (and it was implicated by the original Robot Reply), 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 his potential response that 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 computationalist explanation, that what goes in as hypothesized content into the language acquierer must be interpretable inside, and—this will be the unexpected part for Searle coming from someone who agrees with him—it is true of the child acquiring a language, and the computational system that models the process.

To show that 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:

> "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."

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 see how Keller began to take in content along with the form to give rise to a computable grammar needs a bit of background. I will summarize from her personal recollections, Keller (1905), hereafter *HK*, rather than rely on external observations.

Keller lost her sight and hearing irretrievably at the age of 17 months, when she had already spoken her first words (*soup, water*). By her own admission, she was on her way to become a bitter child because of frustrating failures in expressing herself, when her caretaker Anne Sullivan arrived before her 7th birthday. She eschewed earlier methods and taught her finger spelling. At the end of first day she has two

finger-spelled words, *doll* and *water*. However, she recalls clearly (*HK*:46) that she had difficulty disambiguating *mug* and *water*, although Sullivan was carefeul to run water through her palm as she finger-spelled *water* in the other. On the same page she reports an even more striking recollection: When she was given a new doll in her hand when Sullivan finger-spelled *doll* on the other, she was confused because it was not her usual doll. *HK* reports this incident 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 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 she 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 one because of severed take-in of content. Rapaport argues that she went into a Chinese room, and then get out of 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 talk about a causal link that is necessary in addition to syntax, to give rise to semantics. This link is assumed to be a kind of non-syntactic semantics that links the agent to the *external* world by being in it, but it is according to him "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, because their existence comes for free when we adopt a computationalist explanation, we need to question why she cannot do what she did as a correspondence problem, much like the compiler and the problem of translation I exemplified earlier in §3.2.

There is another way along these lines to understand Keller's enthusiasm and a-ha moment while making a less radical jump about content. She could have moved from the assumption that *doll* was indexical to one where it is referential. 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 mug and water. Then she perhaps entertained the possibility that lots of her earlier indexical assumptions can be referential too. This would also be an a-ha experience. The jump here is not so radical if it has been done before and deprived of later. Recall her surprise when she found out that the doll in her 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 revise it to $doll'$, or maybe, in clearer steps, from $doll'_1$ to $f(doll'_1)$.

The more interesting question is the following: How did she know that what Sullivan does to one hand when finger-spelling in the other one is semantics but not syntax? If she really were in a Chinese room, she couldn't have known that. This is because the conditions of the Chinese room experiment is such that only formal symbols are allowed to enter it. If Rapaport's argument is that she was once in it, then she 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 she would have to invent generating meanings inside. 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 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, and 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, either because it is physically unrealizable for adults (and for "rooms") who are assumed to have passed the Turing test, or 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, for understanding. A hypothesized meaning goes in too. What avoids infinite regress here is the computationalist assumption, that it can't be turtles all the way down,[12] so that those meanings will be interpreted by the virtue of syntax taking them to ground cases where no further syntactic translation takes place.[13]

3.4 Part II summary

The argument that I made about the Chinese Room may appear to vindicate Searle, but actually it does not. What we 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 a more recent statement, Searle 2002:51: "The fundamental claim [of Chinese Room] 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." True. But this is quite pessimistic about the explanatory role of computers, and seems to suggest that they should know their place as instruments. He does not help us understand cognition: "In this science [of consciousness and in-

[12] I first saw this phrase in the sense closest to current discussion in Ross 1967, who attributes it to William James. He recalls it with a "bull's eye relevance to the study of syntax."

[13] Without this assumption we would not be too far from linguistic relativism of the Sapir variety, by 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—if language caused thought—level of inferential ability compared with little (almost no) linguistic exposure up to puberty, such as Genie (Fromkin et al. 1974, Curtiss et al. 1974). Assuming that language is expression of thought seems to avoid these problems. By current argument, it is not only empirically on better grounds, it is also a necessary consequence of thinking that language is a computational mechanism. On the other hand, syntactic semantics requires it as an extra assumption. (But let me stress again: nobody is denying the role of compositional semantics in syntax, the question is whether syntax alone can cause semantics.)

tentionality] the computer will play the same role that it plays in any other science. It is a useful tool for investigation"*ibid*:68.[14]

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 two ends of the correspondence. (Even that is not enough for the computer to have conscious experience. Its job is to explain the process. It seems easier to just drop "(if) computers (still) can (not) do this" kind of rhetoric.) The mechanism suggested would be sufficiently clear if we follow the intensional view. Anyone can peek inside to see if the algorithm is cheating. (One such algorithm is given in Abend et al. 2017.) If it is not possible for the machine to face these circumstances, then it is not possible to computationally 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 simple. 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 proves its worth in assessing Block's argument and Helen Keller's case. She 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 syntactic one's story is well-known from analogies to purely formal processing by a Turing machine, and the semantic one could not have worked by itself, using content all the way, because if it worked it would mean that she could have done this before when she was exposed to the wind, grass, water etc. earlier, but she did not. She needed a medium in which she could solve the correspondence problem herself. Nobody had provided a possible medium for *that* before Sullivan arrived.[15]

[14] Ford 2011:70, who defended his views against Rapaport, is less apathic 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)."

[15] Keller's case is different from the situations of children who are 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 Newport 1995 for a summary.

4 Conclusion

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. It is not necessarily analogical either. The practice tries to make the terms of the mechanism of the correspondence clear so that empirical theories can be built on 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. What makes them a political force is the disproportionate complexity of what *we* can do with their simplicity. Nobody would be afraid of calculators, simple or complex, although we may have worries about them, for example for those that fly airplanes. I think that people implicitly adopt the intensional view of computers, both professionals and non-professionals, when they use computers to rise above data in a correspondence problem.

References

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

Aaronson, Scott. 2013. Why philosophers should care about computational complexity. In *Computability: Turing, Gödel, Church, and beyond*, eds. B Jack Copeland, Carl J Posy, and Oron Shagrir. MIT Press.

Abend, Omri, Tom Kwiatkowski, Nathaniel Smith, Sharon Goldwater, and Mark Steedman. 2017. Bootstrapping language acquisition. *Cognition* 164: 116–143.

ACM. 2012. ACM Turing Centenary Celebration. Association for Computing Machinery, June 15–16, San Francisco.

Bickhard, Mark H. 1996. Troubles with computationalism. In *Philosophy of psychology*, eds. W. O'Donohue and R. Kitchener, 173–183. London: Sage.

Block, Ned. 1978. Troubles with functionalism. In *Minnesota studies in the philosophy of science*, ed. C. Wade Savage. Univ. of Minnesota Press.

Block, Ned. 2002. Searle's argument against cognitive science. In Preston and Bishop (2002).

Bringsjord, Selmer, and Joshua Taylor. 2005. An argument for P=NP. `arXiv:cs/0406056`.

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

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

Cariani, Peter. 1998. Epistemic autonomy through adaptive sensing. In *Intelligent control (ISIC)*, 718–723. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS).

Cockshott, Paul, Lewis M Mackenzie, and Gregory Michaelson. 2012. *Computation and its limits*. Oxford University Press.

Copeland, B Jack. 2002. Hypercomputation. *Minds and machines* 12 (4): 461–502.
Copeland, B Jack, and Oron Shagrir. 2011. Do accelerating Turing machines compute the uncomputable? *Minds and Machines* 21 (2): 221–239.
Curry, Haskell B. 1951/1970. *Outlines of a formalist philosophy of mathematics*. Amsterdam: North-Holland.
Curtiss, Susan, Victoria Fromkin, Stephen Krashen, David Rigler, and Marilyn Rigler. 1974. The linguistic development of Genie. *Language* 50 (3): 528–554.
Dewdney, A. K. 1984. On the spaghetti computer and other analog gadgets for problem solving. *Scientific American* 250 (6): 19–26.
Fodor, Jerry. 1975. *The language of thought*. Cambridge, MA: Harvard.
Ford, Jason. 2011. Helen Keller was never in a Chinese Room. *Minds and Machines* 21 (1): 57–72.
Fromkin, Victoria, Stephen Krashen, Susan Curtiss, David Rigler, and Marilyn Rigler. 1974. The development of language in Genie: a case of language acquisition beyond the critical period". *Brain and language* 1 (1): 81–107.
Gandy, Robin. 1980. Church's thesis and principles for mechanisms. *Studies in Logic and the Foundations of Mathematics* 101: 123–148.
Gleitman, Lila R., and Elissa L. Newport. 1995. The invention of language by children: Environmental and biological influences on the acquisition of language. In *Language: An invitation to cognitive science*, eds. Lila R. Gleitman and Mark Liberman, 1–24. Cambridge, MA: MIT Press. 2nd ed.
Gödel, Kurt. 1946. Remarks before the Princeton bicentennial conference on problems in mathematics. In *Undecidable*, ed. Martin Davis. New York: Raven Press. 1965, p.84.
Graham, Paul. 1994. *On Lisp*. Englewood Cliffs, NJ: Prentice Hall.
Horsman, Clare, Susan Stepney, Rob C Wagner, and Viv Kendon. 2013. When does a physical system compute? *Proc. of the Royal Society A* 470 (20140182).
Hoyte, Doug. 2008. *Let over lambda*. HCSW and Hoytech.
Jay, Barry, and Thomas Given-Wilson. 2011. A combinatory account of internal structure. *The Journal of Symbolic Logic* 76 (03): 807–826.
Keller, Helen. 1905. *The story of my life*. Garden City, NY: Doubleday.
Knuth, Donald E. 1973. Searching and sorting, The Art of Computer Programming, Vol. 3 Addison-Wesley, Reading, MA.
Knuth, Donald E. 1996. *Selected papers on computer science*. Cambridge University Press.
Knuth, Donald E. 2014. Twenty questions for Donald Knuth. `http://www.informit.com/articles/article.aspx?p=2213858`.
Lenneberg, Eric H. 1967. *The biological foundations of language*. New York: John Wiley and Sons.
Lewis, Harry R., and Christos H. Papadimitriou. 1998. *Elements of the theory of computation*. New Jersey: Prentice-Hall. 2nd ed.
Mills, Jonathan W. 2008. The nature of the extended analog computer. *Physica D: Nonlinear Phenomena* 237 (9): 1235–1256.
Newell, Allen, and Herbert Simon. 1976. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM* 19 (3): 113–126.
Pask, Gordon. 1968. Colloquy of Mobiles. ICA London Exhibit.

Piccinini, Gualtiero. 2008. Computers. *Pacific Philosophical Quarterly* 89: 32–73.

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

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

Preston, John, and Mark Bishop, eds. 2002. *Views into the Chinese room: New essays on Searle and artificial intelligence*. Oxford University Press.

Rapaport, William J. 1988. Syntactic semantics: Foundations of computational natural-language understanding. In *Aspects of artificial intelligence*, ed. J. H. Fetzer, 81–131. Kluwer.

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

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

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

Ross, John Robert. 1967. Constraints on variables in syntax. PhD diss, MIT. Published as **?**.

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

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

Searle, John R. 1980. Minds, brains and programs. *The Behavioral and Brain Sciences* 3: 417–424.

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

Searle, John R. 2001. Chinese Room argument. In *The MIT encyclopedia of the cognitive sciences*, eds. Robert A. Wilson and Frank C. Keil, 115–116. Cambridge, MA: MIT Press.

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

Simon, Herbert. 1969. *The sciences of the artificial*. MIT press.

Turing, Alan Mathison. 1936. On computable numbers, with an application to the entscheidungsproblem. *Proc. of the London Mathematical Society* 42 (series 2): 230–265.

Wang, Hao. 1974. *From mathematics to philosophy*. Routledge & Kegan Paul.