Chapter 1
What Is a Computational Constraint?

Cem Bozşahin

Abstract The paper argues that a computational constraint is one that appeals to control of computational resources in a computationalist explanation. Such constraints may arise in a theory and in its models. Instrumental use of the same concept is trivial because the constraining behavior of any function eventually reduces to its computation. Computationalism is not instrumentalism. Born-again computationalism, which is an ardent form of pancomputationalism, may need some soul searching about whether a genuinely computational explanation is necessary or needed in every domain, because the resources in a computationalist explanation are limited.

Computational resources are the potential targets of computational constraints. They are representability, time, space, and, possibly, randomness, assuming that '**BPP** = **BQP**?' question remains open. The first three are epitomized by the Turing machine, and manifest themselves for example in complexity theories. Randomness may be a genuine resource in quantum computing.

From this perspective, some purported computational constraints may be instrumental, and some supposedly noncomputational or cognitivist constraints may be computational. Examples for both cases are provided. If pancomputationalism has instrumentalism in mind, then it may be a truism, therefore not very interesting, but born-again cannot be computationalism as conceived here.

1.1 Introduction

If I am asked to explain starvation in Korea or Hume's billiard balls, I would not say they are because of computation, and, I suspect, neither would an expert. But why not? After all, all the participants in either story are physical entities or their properties, and any physical property is computational if Deutsch (1985, 1989) thesis is right (but read on).

C. Bozşahin (✉)
Cognitive Science Department, The Informatics Institute, Middle East Technical University (ODTÜ), Ankara, Turkey
e-mail: bozsahin@metu.edu.tr

V.C. Müller (ed.), *Computing and Philosophy*, Synthese Library 375,
DOI 10.1007/978-3-319-23291-1_1

Deutsch reformulates Church-Turing thesis using quantum computers, and the examples in the beginning only relate to the revised version of the thesis, that all finitely realizable things are finitely computable, and universal computation is physically realizable, sometimes misleadingly called Church-Turing-Deutsch thesis. Turing and Church talked about effective computation only. Turing did mention physical realizability of effective procedures, but that is far from saying anything physical is computable, as noted by Copeland (2008). (There is further controversy here, one that involves whether Turing addressed humanly effective computations—Copeland interpretation—or effective computations including machines. Hodges (2013) takes the more general view, with which I agree. Otherwise, why would Turing extend his formal model to his imitation game?)

If anything, Turing computability entails physical realizability, but physical realizability does not entail Turing computability. If we have a perfect circle, we have the number π. Unit right triangle gives us $\sqrt{2}$. But there is no effective calculation of $\sqrt[\pi]{2}$. We can question whether there is such a thing as perfect circle or unit right triangle, or whether we should turn to computing over reals and complex numbers. As Cockshott et al. (2012), Deutsch (2011) argue, the question boils down to impossibility of perfect error correction in nondigital computing.

In the world of explanations, for example if DNAs really compute, failure of error correction leads to cancer, and the quest for cure of cancer becomes a search for perfect error correction. The hope very much depends on whether DNAs compute analog way—futile case—or digital, for no measurement is possible without error. If *we* formulate the behavior of DNA as a computational problem, by encoding/decoding hence by having a representation, it is a different enterprise, and some exploits of inherent parallelism is expected in an instrumental way, and, who knows, we might explain cancer if we understand its constraints.

The logic of the argument in the current paper is as follows: we need to ascertain whether we are looking at computation in a process. The intuitions of many computer scientists and physicists about this aspect are nicely put in a framework by Horsman et al. (2013). Once we have that, we need to find out whether a computational explanation is worth considering. If it is, then the degrees of freedom in the data must be formulable in computationalist terms, i.e. its computational constraints must be discernible from functional, cognitive, and other kinds of constraints. This way we can distinguish fundamental constraints, the ones that do the explaining, from auxiliary ones.

Therefore it is important to realize just what is mechanical in computing, and what are its resources. I will cover these aspects first, then define the resources of a computational system as possible targets of constraints, both in a theory and in a model. We then look at purportedly computational constraints, and to noncomputationalist explanations that might benefit from having another look at their domain from this perspective.

1.2 Turing-Deutsch Resources

For reference throughout the paper, let us define a Turing Machine M as TM $M=(Q,\Sigma=\{0,1\},\Gamma=\{0,1\}\cup\{B\},\delta,s,h)$, where Q is a finite set of states, $s\in Q$ is the start state, $h\in Q$ is the halt state, δ is the transition function of a single-tape standard TM (i.e. infinite on one side, just like natural numbers, with a designated start cell), 0s and 1s are input (Σ) and output (Γ) encodings, and B is blank. The starting configuration of a TM for input ax is $(s,\hat{a}x)$, and the halting configuration of a TM is (h,y), where $x\in\Sigma^*, y\in(\Sigma\cup\Gamma)^*$, and $\hat{a}$ designates current tape head looking at symbol a. Using 0s and 1s for any kind of computation is no loss of generality because any string can be converted to its unique Gödel number, and that number has a unique bit string representation with an inverse (i.e. input and output can be translated back to the source vocabulary).

M computes a function $f(x)$ iff $M(x)=f(x)$ for all x, by which, using eta equivalence of lambda calculus, $M=f$. (Read 'computes' as 'decides' if f is a decision procedure.) The function f as a decision process is semi-decidable if there is a TM $M(x)$ for it (it can be represented), which only halts if $M(x)$ is in the language of f ending in h, in which case we interpret h as "yes" state. The Halting Problem is semi-decidable; we can write a TM for it but we cannot decide with it. Some problems are such that there is no TM representation for them, for example '$f=\pi$?' is uncomputable. There is no TM for it with full precision, therefore we cannot even write purported decision TMs such as 'is x the same as π?' let alone semi-decide with them. Therefore *uncomputable* means two things: either we have no way to represent the process as a TM, or, if we do, it is not decidable, but may be semi-decidable. This exhausts the set of undecidable problems.

It is clear that uncomputable problems are not resource-sensitive, but computable ones show gradient use of resources, as reflected in complexity theory.[1] The standard Turing Machine has been extended in many ways, from multiple tapes, two-way infinite tapes, multiple tape heads, to random-access memory. So far no one has managed to extend them by having infinite states in finite computations, but research on related area is ongoing in the form of *hybrid automata*, where a finite-state discrete system is augmented with an analog system of continuous variables, as a proxy for infinity. The impossibility of perfect error correction, like in all analog components, makes them suspect as universal models of computation (unreprogrammability). In the current paper, I suggest that there is another philosophical problem with this route: the auxiliary component would make the whole

[1] Interaction of computational complexity, which looks at resource *behavior* of functions both at the level of theory and the model, and algorithmic information theory, which looks at resource usage at program sizes, not behaviors, in the form of Chaitin-Kolmogorov-Solomonoff complexity, is an open research area; see e.g. Chaitin (1987). There is also the notion of logical resources, i.e. number of variables and quantifiers, and size of a formula, which leads to descriptive complexity. I leave the latter complexity measures to another paper.

system instrumental rather than explanatory. Constraints on it can be of arbitrary nature. Computational constraints, I suggest, are not like that.

One example of transcending the TM is Abramson's (1971) Extended Turing Machine (ETM) that can store a real number in a single cell. We might try to hypothesize that the tape can hold holistic units such as perfect circle (for π), or right unit triangle, to compute $\sqrt{2}$. Abramson machine has two tapes, one for reals, and one for discrete elements. Members of first tape are fetched somehow. But 'how' question is difficult to answer. (If we could be sure of being able to represent the input in full precision in an enumerable way, we would be ensured of finite computation by ETM if the function is calculable as a countable polynomial.) Blum et al. (1989) machines turn the table around by not feeding the real input bit by bit but as a mathematical object. Finite computation is ensured not by step count but by imposing a built-in constant on the result. If certain operations such as finding the greatest integer in a real can be done in constant time, such machines can be more powerful than TMs. If this is realizable, then we have a nontrivial extension of TMs, and its resources would be of great interest.[2]

We can take the Abramson route in such problems, and try to guarantee finite computation by measuring the distance from a desired output of finite precision. (He proves that this is the only way an ETM can tell whether two analytic functions are equal.) This is also tantamount to using the computer as an instrument rather than an explanatory mechanism. Or, to stay within bounds of a computational explanation, we could say that the question relates to difference between complexity, which presumes decidability, and computability, which does not. Then we can question the need for representation for certain parts of the problem, and we can entertain the possibility of having an instrumental mechanism working in tandem with a computational one, precisely because that mechanism is not computational although it may be physically realizable. Language would not be such a problem, but vision might.

Let us consider now the most natural extension of TMs besides multiple tapes and heads: probabilistic TM (PTM). We know that a PTM can be reduced in terms to a nondeterministic TM, which in turn can be reduced to a standard TM using multiple tapes. (These tapes make it convenient to find out that every nondeterministic computation of say $f(x)$ ends with the same value in $M(x)$, i.e. it is a function.) PTM's complexity class **BPP** has the property **BPP** $\subseteq$ **BQP**, but

[2]In this sense, the notion of representation is not vague in a computationalist explanation, as it is sometimes claimed in dynamical approaches to cognition, e.g. Beer (2003). It is something that makes other resources of a TM accessible. Its ontology in a model is a related but different practice. Reference by an internal representation can be wrong at times, as in early child language acquisition, but representation itself is robust. Of course, not all dynamicists would agree on uselessness of a representation, but complexity results from dynamical systems, such as the one currently discussed in Blum et al. (1989), must already alert the dynamicist and embodied cognitivist that computation in cognition was never parodied as rule-following only; it's about management of computational resources; see Mirolli (2012), Valiant (2013) and Aaronson (2013b) for some discussion.

the reverse containment is not known, therefore randomness might play different roles in PTMs and Quantum Turing Machines (QTMs). It might be a resource in a QTM, whereas it can be translated to other resources in a TM. These are complexity classes, and they do not add new computable functions to our repertoire. From a resource perspective, however, they might make a difference by adding randomness such as in quantum computing.

1.3 Trans-Turing Resources

Consider another extension in this regard, the Putnam-Gold (1965) machine, PGTM. Such machines record their output on a separate tape, whenever they can. (It is easier to think of them as a 3-tape TM, one holding the input TM, the other the input to input TM, and the last one the output.) For example, to attempt the Halting Problem of TM M, a PGTM writes 'no' on the output tape before it executes M, and changes it to 'yes' if M halts. If it does not, we still have an answer for the object machine M, but the PGTM might run forever. This manouvre appears to add more problems to the class of computable functions, but it only delays the infinite regress by one more step: the halting problem of PGTM is not formulable. If the current answer is 'no', it only means it has not halted yet, which is not an answer to the halting problem of PGTM. This way of thinking is not going to add more resources to computing, as the reduction of multiple tape TMs to standard TMs already implicated.

Looking at the output tape without worrying about the status of the object machine on the other tape is same as having a finite but indefinite precision. Assume that we ask the question 'is $x=\pi$?' to a PGTM. The answer can be 'yes', which means the indices are the same *so far*. 'No' as a tentative answer might mean the indices examined are not the same, and we are done. As a *final* answer it would mean PGTM knows the value of π, which is not possible. Uncomputable problems in kind remain uncomputable in PGTM.

Appeals to infinite resources to transcend Turing computability are not facing merely a technological problem; they seem to lack a physical substrate on which to execute an algorithm. Take for example hypercomputation by speed doubling at every step (or halving the amount it takes to reach the next step; Copeland 2002). We either exceed the speed of light in some problems, such as computing $x \stackrel{?}{=} \sqrt{2}$, or require infinite energy to keep it going (Cockshott et al. 2012). Special Theory of Relativity and thermodynamics have shown that these are not possible outcomes of physical events. The notional extension has no realizability.

There are also examples where rethinking purportedly computational constraints as something else, and vice versa, might lead to a new understanding of the phenomenon under study. In the end the question arises about the choice of automata being the right level to call a constraint computational.

Membrane computing is one such area where the primitives of the executive substrate, biology, seems so remote compared to a TM. (This did not stop Turing

from inaugurating morphogenesis.) Such systems perform synchronous transitions from states to states, therefore they have configurations, i.e. they are computing mechanisms. Păun and Rozenberg (2002) show that halting configurations of membranes exist, and the system has complexity classes. Inherent parallelism is kept under control by permeability, which is control of trading time for space to solve exponential-time problems in polynomial time but in exponential space. Unsurprisingly, it can be translated in terms to space (hence resource) control. This can be explained by Turing computation's properties $\mathbf{NP} \subseteq \mathbf{EXP} \subseteq \mathbf{EXPSPACE}$.

In summary, the Turing machine seems to rise above normativity of definitions; it seems to manifest a natural boundary for computability. Even Gödel, who was deeply suspicious of taking formal definitions too seriously, has acknowledged this fact: "with this concept [TM] one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e. one not depending on the formalism chosen." (Gödel 1946).[3]

This result, and Gödel's point about the epistemological aspects, bring computational explanation as a genuine device in science making. A natural consequence is to look at what these explanations can bring in as a resource, to posit a computational constraint on a phenomenon as an epitome of this way of thinking.

Ever since Turing, these resources have been thought to be mechanical, and it is important to be explicit about what 'mechanical' means. Human computers of Turing did their work "mindlessly," to the point of full substitutability by another human. His imitation game extends the experiment of same kind of computation to artificial computers, and it is clear that the whole process depends crucially on one thing: having a representation. Without representation, there is no need to encode a process in the beginning, to start the physical substrate to compute, and no need to decode at the end, to report it back. These representations live in their own abstract world, in the theory and in the models predicted by the theory. For example the notion of making more effort in a computation translates to taking more steps in a Turing model, with measures in abstract time and space, as we do in complexity classes $\mathbf{P}, \mathbf{NP}, \mathbf{SPACE}$ and others.

Trans-Turing alternatives to computation carry the implication that physical processes might compute, but not necessarily the Turing way. We expect some measurement in such machines, with error control, but that also applies to standard TM, as finite precision of representation. Measurement error cannot be zero. In digital computing it is *made* zero after every sampling, and errors never accumulate. It leads to robust representability. Therefore error correction is not an engineering kludge, it is the fundamental principle of digital processes.[4]

[3] Also note that this comment comes from a man who had considered the mind connection of Turing unthinkable, assuming that Turing flawed by considering possible states of the mind to continue to be finite when the mind changes (Wang 1974: 325, quoting Gödel). $\mathbf{P} \neq \mathbf{NP}$ conjecture brought another perspective to the mind problem; see Aaronson (2013a,b).

[4] As an example from cognitive science of language, consider the word *sleep*. In the spectrum of sleep-like thoughts and behaviors, one ceases to use the word 'sleep' if it strays too far from someone's understanding, and from desires of communication. This is self-imposed error

Once represented, the number π can be calculated to any index by a standard TM, but we still cannot entertain questions such as 'is π same as what is measured on the nondiscrete tape?' The notional extensions must show they are physically (but not necessarily technologically) realizable somehow, when they claim to compute trans-Turing functions. This is because a computational constraint, whatever it is, can be imposed *during* the execution of the function, i.e. in a model, not in the abstract formalism, but also *before* the execution of the function, in the abstract theory, which then leads to models which are by definition equipped with the constraint. For example choosing finite-automata in theory affords only certain kinds of computation in its models.

1.4 Computationalist Explanation

Ascribing computational powers to everything has been called pancomputationalism (Floridi 2004; Piccinini 2003, 2007, and early Putnam 1967; see Müller 2009). For what I have in mind, 'born-again computationalism' is probably a better term, which is the doctrine which says belief in computation brings out an explanation for everything. Not quite, as the previous argument shows (see also Horsman et al. 2013).[5] Pancomputationalism may be a truism if Deutsch is right, but born-again computationalism cannot be. It also follows that avoiding a computational explanation when there is one can be equally blindfolding. That relates to the nature of computational constraints, the main subject of this paper.

Apparent degrees of freedom in so-called new kinds of computation seem to proliferate, which makes it harder to see the following: if we stop at some level which is considered computational in some conception, say computations of

correction, which avoids uncountably many sleep-related words. In that sense language is a digital process, i.e. it is discrete and robustly representational (Haugeland 1997), and words are its proofs.

[5] A cogent warning about why pancomputationalism may be a truism is given by Horsman et al. (2013: 20): "A common, and unfortunate, method of ascribing computational ability to a nonstandard system is as follows. A novel computing substrate is proposed (a stone, a soap bubble, a large interacting condensed-matter system, etc.). The physical substrate is set going and an evolution occurs. At a certain point, the end of the process is declared and measurements taken. The initial and final states of the system are compared, and then a computation and a representation picked such that if the initial state, and final states are represented in such a way then such a computation would abstractly connect them. The system is then declared to have performed such a computation: the stone has evaluated the gravitational constant, the soap bubble has solved a complex optimization problem and so on.

Such arguments, without any further testing or evidence, are rightly treated with suspicion. Given the vast range of representation for physical systems available, almost any computation can be made to fit the difference of initial and final physical states of a system. If such arguments really were correct, we would not only have to conclude that everything in the universe computes, but that everything computes every possible computation all of the time. Such extreme pancomputationalism is even less useful than the usual kind."

individuals in social computing, are we looking at a computational explanation? In cases such as Simon (1996) where power laws and resource boundedness of individuals explain what is socioeconomically organizable, we probably do. I know of no other way of narrowing down the degrees of freedom except to appeal to theoretical limits of what is being claimed and its physical possibility, and to their differences from the basic definition we accept in computer science, namely the Turing model of computation[6]:

> In general, a demonstration that a new system is more powerful than a Church-Turing system involves showing that while all terms of some Church-Turing system can be reduced to terms of the new system, there are terms of the new system that cannot be reduced to terms of that Church-Turing system; in other words, the new system has greater expressive power than that Church-Turing system and hence of any Church-Turing system. Incidentally, it would be truly astonishing if a new system were demonstrated whose terms could not be reduced to those of a Church-Turing system; that is, the new system and Church-Turing systems had incomparable expressive power. Cockshott et al. (2012: 176)

The next section enumerates the terms of a Turing system.

1.5 Computational Constraint

We can distinguish computation as an auxiliary mechanism in an explanation, and computation as the explanatory mechanism. In the first case we need not talk about computational constraints because lifting or imposing the constraint would not make a difference in the explanation. It might, however, constrain an auxiliary function. Conceived that way, any function f that we can execute in a universal TM acts as a constraint: if the problem requires f to be computed, then computing some $g \neq f$ would not serve the purpose. We must have f. This is instrumental use of a computer, i.e. experimentation by programming, and f as a constraint would not sieve out impossible outcomes of the theory.

As an explanatory mechanism, a computational constraint can appeal to resources of computation. As discussed before, these resources can reveal themselves in the theory, in which case the constraint would be embedded in every model arising from the theory, to the extent that, looking from the model's side, it might be difficult to see the constraint. One such example is finite-state machines, which all carry the TM constraint that the machine makes no turns in the tape. We can execute finer resource control in theory than Chomsky hierarchy, by for example employing

[6]Physical possibility is crucial because every computation requires an executive medium. This is one difference between functionalism and computationalism, where in the latter the primitives of the executive substrate may shape the form of the computed function. For example, membrane computing's *in* command (to permeate a local membrane) has a Turing correlate (Păun 2000), but we would expect membrane models to be programmed more naturally with *in*. Making a difference in complexity classes makes a difference in computationalism, but not necessarily in functionalism.

one-turn PDAs to get linear languages, and finite-turn PDAs to move to ultralinear languages (Harrison 1978), before we reach context-freeness. Once realized in a model, these constraints are no longer explicit but their effects are built in.[7]

Other computational resources are revealed on the model side: time, space, and randomness (because whether **BPP** = **BQP** is not known). The first two are exploited in complexity theory. The last one is not a resource per se in classical computing, as we have seen. It might be a resource in quantum computing and oracles, because if we are satisfied with a truly random answer, then such machines can provide one without effort. This is not the role of randomness in quantum computing, because if it were, we would have to take a wrong answer which might be spit out randomly as the result of a quantum computation. QTMs do compute functions; they can be programmed in principle. Randomness has unsettled consequences in computer science; for example it is tempting to think of **P** to be a proper subset of **BPP**, but it may in fact be the case that **P** = **BPP** (see Arora and Barak 2009 for discussion).

If a machine is not required to give a causal explanation of how the answer was constructed, it can reduce the amount of work to nil in principle. This is what oracles do. Quantum machines, however, must be able to provide an explanation because of superposition and unitary transformations (reversible computing).[8] By playing with randomness that we can tolerate, we can control the amount of work a computational system is expected to do. That makes randomness a resource, and a potential target for a computational constraint. This is true of standard TM computation too, but not as much as the quantum variety: interactive proofs use randomness of the verifier as a resource in finite number of interactions between the prover and the verifier (the prover can be probabilistic too, without change of results). The class **IP** which characterizes such proofs is in **PSPACE**, i.e. we do not need more than efficiently space-bounded computations to deal with such kind of randomness, hence its constraints may have already been spoken for in terms of space.

1.6 Computational Constraint Versus Instrumental and Cognitivist Constraints

Ecorithm is a term Valiant (2013) uses for nature's algorithms for learning and prospering in complex situations, and it might at first sight appear to suggest some form of born-again computationalism. This would be an incorrect interpretation.

[7] Notice that turn itself is not a resource. It translates to bounded use of space, which is a resource.

[8] Roughly speaking, here is how we can get the answer algorithmically. Assume that we expect QTM M to compute $M(x)=f(x)$, given x. Input is prepared in a superposition by for example a series of Hadamard transformations, from $|0\rangle$ to $|x0\cdots0\rangle$. M computes and ends in a state $|xf(x)0\cdots0r\rangle$, where r is the residue of quantum gates. Copy—not clone—the result to unused bits to get $|xf(x)f(x)r\rangle$ by a series of quantum operations $|b_1b_2\rangle \mapsto |b_1(b_1 \oplus b_2)\rangle$. Then run the gates in reverse to get $|x0\cdots0f(x)0\cdots0\rangle$, and Hadamard to get $|0\cdots0f(x)0\cdots0\rangle$, viz. $|f(x)\rangle$.

These algorithms are not computational because they are natural, they are natural because they are resource-sensitive in a computational way, in this particular case by complexity measures and sampling. These constraints are explanatory, and not pancomputationalist. They do assume that the problem becomes formulable in computationalist terms if there is an encoding/decoding mechanism of its instances, much like in the sense described by Horsman et al. (2013).

Some cognitive constraints may turn out to be computational as well. (What I am suggesting is that, a computationalist reformulation that can posit computational constraints only, to stay at this level of explanation, may be an alternative explanation to cognitivism. In the instrumental sense, any constraint eventually becomes a computer-implemented function, as argued earlier.)

Consider one example. Gentner (1982) claimed that children acquire nouns first universally, because nouns are names for things, and there are things around when a child grows. Framed as such, this is a conceptual (cognitive) constraint or bias, toward the objects and their perception, called Natural Partition Hypothesis. A computationalist alternative is to suggest that maybe short, unambiguous and frequent things are learned first, whatever they are, because these aspects can be shown to reduce computational effort (see Bozşahin 2012 for more discussion). For example, contra natural partition hypothesis, Mandarin children seem to show no cognitive bias toward nouns (Tardif 1996). It may be because Mandarin words are uniformly short. Tzeltal children use verbs very early (Brown 1998), which would not be surprising from a computationalist perspective because these verbs are argument-specific therefore unambiguous (e.g. Tzeltal has a generic word for 'eat' and a specific word for eating tortillas, and children acquire the specific ones first). Turkish children seem to use parts of speech equally frequently (Avcu 2014). It might be simply because they are equally frequently exposed to them through morphology (e.g. clauses can be nominalized to bear nominal inflection, and nominals can be predicates, requiring verbal inflection). Of course this simplistic scenario cannot be the whole story extending to adult life, but keep in mind that this is just one computational constraint. Expressivity and the need to communicate will steer the child in somewhat contrary directions. (We can be very creative and use personal words all the time. That would be very expressive but highly uncommunicative. We can be very communicative by using no novel word forms at all, but that would not be very expressive, etc.) This too can be shown to be a computational constraint: in artificial simulations of lexicon development, finite convergence to common vocabulary requires control of synonymy, and indirectly, expressivity and communicative success (Eryılmaz and Bozşahin 2012). In particular, full communicative success implies very early convergence, and full expressivity implies no convergence.

Consider another example. Tomasello's gradual transition to a position where chimpanzees are considered to have a relatively simple mind (from the earlier position of having no mind at all), suggests a potential for computational explanation as well (Tomasello and Call 1997; Tomasello et al. 2003). Not surprisingly, he and his colleagues formulated the constraints in terms of cognitive capacities, joint attention, communicative intent and others. But, there seems to be a continuum

along the computational resources. Chimpanzees are known to make plans (see Steedman 2002 for a review, and Jaynes 1976 for more examples). But they seem to have difficulty going from "I intentions" to "we intentions," to use Searle's (1990) terminology (a reliable source of mind anti-computationalism). We can conceive the instrumental plans of chimpanzees as stack of actions, and collaborative we-intentions of humans as stack of stacks (Bozşahin 2014). We can prove that they make fundamentally different uses of computational resources. We can then go back and try to explain chimpanzee's awareness of other minds in computationalist terms, perhaps with more experiments to tell computational constraints apart from cognitive ones.

There is an area where we know that computationalist explanations can predict an upper bound on the nature of complexities: dependency structures in language are known to be over and above context-freeness (Shieber 1985). And how much above has a computational answer: linear-indexed dependencies, therefore linear-indexed languages. This computational constraint in theory translates itself to embedded PDA in models (Vijay-Shanker 1987) (a stack of stacks, hence the appeal to computational continuity above), which applies to all the models without the need of an explicit mention. CCG and TAG are two such theories.

In contrast, we can have another look at explanations that seem to use computational terminology but not computational resources. Take for example the so-called functional constraints on language, the kind Hawkins (1994) identifies as some word orders causing "processing difficulties" because they purportedly make it difficult to connect earlier constituents in parsing, which is supposedly a function, to mother nodes in a tree of constituents. Judging from the terminology, it appears to be a computational constraint. But no automata class is singled out by this constraint, or a resource. The theory itself is not constructed in a grammar formalism that spells a certain class of automata either (finite-state, push-down, embedded push-down etc.). We have no grounds to see what kind of computations is left out by the constraint, and we are left with an impression that parsing is a performance function. However, children parse to learn, rather than learn to parse (Fodor 1998), and there are computational constraints on the process. For example, Joshi's (1990) constraints are computational in this sense. A truly functional constraint would not be computational. Hawkins (1994) appears to posit a functional constraint, and instrumental, rather than computational.

Automatic computations and interactive ones can be compared from this perspective as well. Turing (1936) called the first kind a-machines and the second c-machines (c for 'choice'). Oracles (o-machines; Turing 1939) differ from both. They need not specify an internal i.e. computational mechanism although they might have one.

Consider the difference between calculating an integral within an interval and playing backgammon. There is no external element in the first one that would force the symbols of its "computations" to communicate with the computer and/or the outside world. For backgammon there is an external element, the dice. The dice throw, however, can be incorporated into a-machine configurations, say by having another tape to enumerate sequence of dice throws long enough to serve any finite

game. If this a-machine must decide whether the same configuration reached several times in a game warrants continuation, asking this question puts a computational constraint on the game, unlike asking for a dice throw, because the automata class changes from a-machines to c-machines, with concomitant results such as potential undecidability. As an open system in this sense, it is constrained descriptively by whatever is delivered by the dice, and computationally by an interaction. For integral calculation, if e.g. switching from integers to reals could change the complexity of the calculation, then we would consider the integer/real constraint a computational one. Blum et al. (1989) suggest that rethinking computation with reals reveals new complexity classes such as **NP**-R of decision problems for R, where R=$\mathbb{R}$ or R=$\mathbb{Z}$. **NP** is same as **NP**-$\mathbb{Z}$, but not **NP**-$\mathbb{R}$.

1.7 Conclusion

If a constraint is truly computational, we can see a computationalist explanation behind it. If not, the constraint may be instrumental. If some phenomenon begs for a computationalist look, there will be telltale signs, most notably parsimonious use of a computational resource, i.e. a Turing-reducible resource: representation, time, space, and randomness (assuming **BPP** $\supseteq$ **BQP** is unresolved, and perhaps unlikely to be true; whether **BPP** $\subseteq$ **BQP** containment is proper is not known).

I have given examples of two kinds: purportedly computational constraints which are not translatable to these terms, and supposedly noncomputational or cognitivist constraints, which might. Different vocabulary does not seem to change this aspect (as we observed in membrane computing), but may make it more opaque or indirect to see the potentially computational nature of a problem.

Computational explanations face the same risk, not least of which ranges from reading Church-Turing thesis stronger than necessary, to pancomputationalism and born-again computationalism.

Acknowledgements Thanks to Mark Addis, Tarek Besold, Mark Bishop, Robin Hill, Doukas Kapantaïs, Stasinos Konstantopoulos, Giacomo Lini, Halit Oğuztüzün, Deniz Şener, Mario Verdicchio, Ioannis Votsis, Aziz Zambak for discussion and advice on the paper and related topics. I am to blame for all errors and misunderstanding.

References

Aaronson, S. (2013a). *Quantum computing since Democritus*. Cambridge: Cambridge University Press.

Aaronson, S. (2013b). 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.

Abramson, F. G. (1971). Effective computation over the real numbers. In *IEEE 54th Annual Symposium on Foundations of Computer Science*, Los Alamitos (pp. 33–37).

Arora, S., & Barak, B. (2009). *Computational complexity: A modern approach*. Cambridge: Cambridge University Press.

Avcu, E. (2014). *Nouns-first, verbs-first and computationally easier first: A preliminary design to test the order of acquisition*. Unpublished master's thesis, Cognitive Science department, Middle East Technical University (ODTÜ), Ankara.

Beer, R. D. (2003). The dynamics of active categorical perception in an evolved model agent. *Adaptive Behavior, 11*(4), 209–243.

Blum, L., Shub, M., & Smale, S. (1989). On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society, 21*(1), 1–46.

Bozşahin, C. (2012). *Combinatory linguistics*. Berlin/Boston: De Gruyter Mouton.

Bozşahin, C. (2014). Natural recursion doesn't work that way: Automata in planning and syntax. In V. C. Müller (Ed.), *Fundamental issues of artificial intelligence (synthese library)*. Berlin: Springer.

Brown, P. (1998). Children's first verbs in Tzeltal: Evidence for an early verb category. *Linguistics, 36*(4), 713–753.

Chaitin, G. J. (1987). *Algorithmic information theory*. Cambridge/New York: Cambridge University Press.

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

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

Copeland, B. J. (2008). The Church-Turing thesis. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy*. Stanford: Stanford University.

Deutsch, D. (1985). Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences, 400*(1818), 97–117.

Deutsch, D. (1989). Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences, 425*(1868), 73–90.

Deutsch, D. (2011). *The beginning of infinity: Explanations that transform the world*. New York: Penguin.

Eryılmaz, K., & Bozşahin, C. (2012). Lexical redundancy, naming game and self-constrained synonymy. In *Proceedings of the 34th Annual Meeting of the Cognitive Science Society*. Sapporo.

Floridi, L. (2004). Open problems in the philosophy of information. *Metaphilosophy, 35*(4), 554–582.

Fodor, J. D. (1998). Parsing to learn. *Journal of Psycholinguistic research, 27*(3), 339–374.

Gentner, D. (1982). Why nouns are learned before verbs: Linguistic relativity versus natural partitioning. In S. A. Kuczaj II (Ed.), *Language development, vol.2: Language, thought and culture* (pp. 301–334). Hillsdale: Lawrence Erlbaum.

Gödel, K. (1946). Remarks before the Princeton bicentennial conference on problems in mathematics. In M. Davis (Ed.), *Undecidable*. New York: Raven Press. (1965, p. 84)

Gold, E. M. (1965). Limiting recursion. *Journal Symbolic Logic, 30*, 28–48.

Harrison, M. (1978). *Introduction to formal language theory*. Reading: Addison-Wesley.

Haugeland, J. (1997). What is mind design? In J. Haugeland (Ed.), *Mind design II* (pp. 1–28). Cambridge: MIT Press.

Hawkins, J. A. (1994). *A performance theory of order and constituency*. Cambridge: Cambridge University Press.

Hodges, A. (2013). Alan Turing. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy*. Stanford: Stanford University.

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

Jaynes, J. (1976). *The origin of consciousness in the breakdown of the bicameral mind*. New York: Houghton Mifflin Harcourt.

Joshi, A. (1990). Processing crossed and nested dependencies: An automaton perspective on the psycholinguistic results. *Language and Cognitive Processes, 5*, 1–27.

Mirolli, M. (2012). Representations in dynamical embodied agents: Re-analyzing a minimally cognitive model agent. *Cognitive Science, 36*(5), 870–895.

Müller, V. C. (2009). Pancomputationalism: Theory or metaphor? In *The relevance of philosophy for information science*. Berlin: Springer.

Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences, 61*(1), 108–143.

Păun, G., & Rozenberg, G. (2002). A guide to membrane computing. *Theoretical Computer Science, 287*(1), 73–100.

Piccinini, G. (2003). *Computations and computers in the sciences of mind and brain*. Unpublished doctoral dissertation, University of Pittsburgh.

Piccinini, G. (2007). Computational modelling vs. computational explanation: Is everything a Turing Machine, and does it matter to the philosophy of mind? *Australasian Journal of Philosophy, 85*(1), 93–115.

Putnam, H. (1965). Trial and error predicates and the solution of a problem of Mostowski. *Journal Symbolic Logic, 30*, 49–57.

Putnam, H. (1967). Psychological predicates. In *Art, mind, and religion* (pp. 37–48). Pittsburgh: University of Pittsburgh Press.

Searle, J. R. (1990). Collective intentions and actions. In M. E. P. Philip, R. Cohen Jerry, & L. Morgan (Ed.), *Intentions in communication*. Cambridge: MIT Press.

Shieber, S. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy, 8*, 333–343.

Simon, H. A. (1996). *The sciences of the artificial* (3rd ed.). Cambridge: MIT press.

Steedman, M. (2002). Plans, affordances, and combinatory grammar. *Linguistics and Philosophy, 25,* 723–753.

Tardif, T. (1996). Nouns are not always learned before verbs: Evidence from Mandarin speakers' early vocabularies. *Developmental Psychology, 32*(3), 497–504.

Tomasello, M., & Call, J.(1997). *Primate cognition*. New York: Oxford University Press.

Tomasello, M., Call, J., & Hare, B. (2003). Chimpanzees understand psychological states—the question is which ones and to what extent. *Trends in Cognitive Sciences, 7*(4), 153–156.

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. (1939). Systems of logic based on ordinals. *Proceedings of the London Mathematical Society, 2*(1), 161–228.

Valiant, L. (2013). *Probably approximately correct: Nature's algorithms for learning and prospering in a complex world*. New York: Basic Books.

Vijay-Shanker, K. (1987). *A study of tree adjoining grammars*. Unpublished doctoral dissertation, University of Pennsylvania.

Wang, H. (1974). *From mathematics to philosophy*. London: Routledge & Kegan Paul.