Philos. Technol. (2013) 26:31–60
DOI 10.1007/s13347-011-0061-4

TARGET ARTICLE

Information Processing as an Account of Concrete Digital Computation

Nir Fresco

Received: 30 June 2011 / Accepted: 26 December 2011 / Published online: 19 January 2012

Abstract It is common in cognitive science to equate computation (and in particular digital computation) with information processing. Yet, it is hard to find a comprehensive explicit account of concrete digital computation in information processing terms. An information processing account seems like a natural candidate to explain digital computation. But when 'information' comes under scrutiny, this account becomes a less obvious candidate. Four interpretations of information are examined here as the basis for an *information processing* account of digital computation, namely Shannon information, algorithmic information, factual information and instructional information. I argue that any plausible account of concrete computation has to be capable of explaining at least the three key algorithmic notions of input, output and procedures. Whist algorithmic information fares better than Shannon information, the most plausible candidate for an information processing account is instructional information.

Keywords Concrete digital computation · Turing machines · Algorithmic information · Shannon information · Factual information · Instructional information · Cognitive science · Algorithm · Program

1 Introduction

It is often assumed, particularly in cognitive science discourse, that computation can freely be described as information processing. The Encyclopedia of Computer Science states that "information processing might, not inaccurately, be defined as 'what computers do'" (Ralston 1976: p. 647). This definition is put to the test in this paper, dealing with the question whether concrete digital computation (i.e. digital computation as it is actualised in *physical systems*) can be adequately explained solely in terms of information processing.

The resulting information processing (IP) account hinges on the particular interpretation of information. It can be interpreted semantically (as factual or instructional

N. Fresco (✉)
School of History and Philosophy, University of New South Wales, Sydney, Australia
e-mail: Fresco.Nir@Gmail.com

information) or nonsemantically (e.g. as Shannon information or algorithmic information). To set the stage, the processing of information is characterised here as the production of new information, its modification or the removal thereof. To a first approximation, this characterisation seems to accord with the operations of a Turing machine (hereafter, TM) reading from and writing to a tape and changing states. Also, intuitively, it seems to describe the practical use of information by natural cognitive agents beyond their mere communication of information.

As suggested above, the focus of the paper is on concrete digital computation rather than abstract computability. I argue elsewhere that formalisms of computability (such as TMs or the lambda calculus) may indeed provide the mathematical tools required for determining the plausibility of *computational level theories* (in the spirit of David Marr's tripartite analysis). Yet, it is concrete computation that is most relevant to the study of cognition as a natural phenomenon (Fresco 2011). Since our interest here is in physical computing systems, human-engineered computing systems such as conventional digital computers and (where applicable) discrete neural nets[1] are used in this paper to exemplify various points. These examples do not limit the generality of the arguments put forward. Human-engineered computing systems can simply be used as paradigmatic cases of computation.

The paper proceeds as follows. I begin in the next section by introducing four possible interpretations of information: Shannon information, algorithmic information, factual information and instructional information. Subsequently, in Section 3, I examine the resulting IP accounts depending on the interpretation of information adopted and explicate the key requirements for a physical system to perform nontrivial digital computation in terms of IP. In Section 4, I argue that any plausible account of concrete computation has to be up to the task of explaining at least the three key algorithmic notions of input, output and procedures. I then examine some problems for the resulting IP accounts concluding that an account based on algorithmic information fares better than one based on Shannon information. Still, only an IP account that is based on instructional information seems to be suitable for individuating nontrivial digital computing systems. Section 5 proposes a plausible IP account based on instructional information. In Section 6, I show how this proposed account deals with problems faced by the other interpretations of information.

2 Semantic Information, Nonsemantic Information and Data

The roots of the conflation of information and computation are in the attempt to explain the mind in the mid-twentieth century. This venture led to a fusion of information-theoretic language, notions of control and TMs. At that time, the

[1] Graham White suggested extending the use of the single-input single output transducer paradigm. One way of going about it is to refer to computing agents instead (i.e., in terms of goals, goal-triggers and actions to achieve these goals). Supposedly, this paradigm would encompass conventional digital computers as well as neural nets. After all, neural nets are arguably the paradigmatic case for parallel distributed processing of information. But this requires significant conceptual groundwork to determine the extent to which computing agents are goal-driven rather than rule-driven. Besides, it is an entirely different matter whether all neural nets compute, and even if they do, whether they perform nontrivial *digital* computation. I discuss this last question elsewhere (Fresco 2010).

information-theoretic language used was based on the Shannon/Wiener cybernetic conception of information. The original ambition of cybernetics was to explain the behaviour of both living organisms and human-engineered machines using a unified theory utilising concepts of control and information (Cordeschi 2004: p. 186). Norbert Weiner, for instance, defined a machine as "a device for converting incoming messages into outgoing messages…. [or] as the engineer would say… [it] is a multiple-input multiple-output transducer" (1966: p. 32). Still, in general, the problems under consideration were based on single-input single-output transducers without any long-range statistical dependencies of transmitted messages.

However, our modern concept of information is broader and need not be limited only to this cybernetic conception. To some degree, everything can be described in information-theoretic terms: from the movements of atoms and molecules through economics, politics and fine arts to ethics and human nature. So broad is the concept of information, that some readings of information unavoidably lead to pan-informationalism. Moreover, there is now even a dedicated discipline within philosophy devoted to study information, namely the philosophy of information (Floridi 2011).

Furthermore, some draw a distinction between 'data' and 'information'. The Encyclopedia of Computer Science (Ralston 1976: p. 641) defines *data* as "physical symbols used to represent information for storage, communication or processing". *Information*, on the other hand, is defined as "knowledge, especially as it provides people (or machines) with *new* facts about the real world (ibid, italics original). Data apparently are the vehicle that conveys meaningful (or semantic) information and could be analysed, for example, by Shannon's information theory (henceforth, SIT) or by algorithmic information theory (hereafter, AIT) discussed below.

In what follows then, my focus on information is confined to how Information *Processing* pertains to concrete digital computation. But in order to understand *how* information is processed, we first need to understand *what* is being processed.[2] Four relevant readings of information, which seem to be potential candidates for an IP account of concrete computation, are explored here with a distinction made between semantic and nonsemantic information.[3]

2.1 Nonsemantic Information Version 1: Shannon Information

Claude Shannon (1948: p. 1) attempted to solve the "fundamental problem of communication": finding the optimal manner by which messages from a source of information are exactly or approximately reproduced at their destination (Piccinini and Scarantino 2011: p. 19). According to Wiener (1948: p. 61), one of the simplest unitary forms of information is the recording of a choice between two equiprobable basic alternatives.

[2] The functionalist approach of ignoring the internal constitution of information and only concentrating on its processing instead is not so useful in our case. The conceptual analysis undertaken in this paper results in different IP accounts depending on what information is taken to be. It also has implications for the applicability of the resulting IP account to concrete computation.

[3] In a similar vein, Gualtiero Piccinini and Andrea Scarantino analyse three potential candidates for an IP account: SI, natural semantic information and non-natural semantic information (2011). They conclude that digital computation does not entail the processing of either SI or natural semantic information and it also need not be the processing of non-natural semantic information.

However, Shannon information (SI) does not entail any semantic content or meaning. SIT has abstracted from the physical media of communication (e.g. the physical composition of the communication channel) so that any relevant physical constraints can be analysed separately. Shannon provided a statistical definition of information as well as general theorems about the theoretical lower bounds of bit rates and the capacity of information flow. His theory approaches information syntactically: whether and how much (not what) information is conveyed (Floridi 2008: p. 119). SI is different from the ordinary usage of 'information'; it tells us nothing about the usefulness of or interest in a message. The basic idea is coding messages into a binary (or any other) system at the bare minimum number of bits we need to send to get our message across in the presence of noise.

Even in this sense, the amount of information conveyed is as much a property of our own knowledge as anything in the message. If we send the same message twice (a message and its copy), the information in the two messages is not the sum of that in each. Rather the information only comes from the first one. Receiving a message may change the recipient's circumstance from not knowing what *something was* to knowing what *it is* (Feynman 1996: pp. 118–120). The more possible messages a recipient could have otherwise received, the more "surprised" the recipient is when it gets that particular message. The average amount of uncertainty or surprise of the recipient is also known as informational *entropy* (Floridi 2009: p. 34).

2.2 Nonsemantic Information Version 2: Algorithmic Information

AIT, which was developed by Andrei Kolmogorov, Ray Solomonoff and Gregory Chaitin, deals with the informational complexity of data structures (but not with the runtime complexity of algorithms). It formally defines the *complexity* or the *informational content* of a data structure, say, a string, as the length of the shortest self-delimiting program[4] producing that string as output on a universal TM (UTM; Chaitin 2003: p. 157, Dunn 2008: p. 590). The algorithmic information (AI) complexity of any computable string (in any particular symbolic representation) is the length of the shortest program that computes it on a particular UTM and halts.

Moreover, Chaitin proposes thinking of a computing system as a decoding device at the receiving end of a noiseless binary communications channel (Chaitin 2003: p. 157). Its programs are thought of as code words and the output of the computation as the decoded message. The programs then form what is called a 'prefix-free' set so that successive messages (e.g. procedures) sent across the channel can be distinguished from one another. Still, he acknowledges that AI has precisely the formal properties of the entropy of SI.

AI may be deemed a competing notion of SI by allowing us to assign complexity values to individual strings and other data types. Whilst SIT analyses the amount of information of a message relative to a group of messages, AIT analyses the complexity of a string as a single message (Adriaans 2008: p. 149). The relative frequency of the message (which is the focus of former theory) has no special import. The length

[4] The domain of each UTM is self-delimited much like programming languages. For they provide constructs that delimit the start and end of programs. A self-delimiting TM does not 'know' in advance how many input symbols suffice to execute the computation (Calude 2002: pp. 34–35).

of the shortest program producing this message is minimal within an additive constant that encapsulates the size of the particular UTM[5] representing the amount of information in that message (Calude et al. 2011: p. 5671).

2.3 Semantic Information

The most common informal analysis of semantic information over the last three decades characterised it in terms of data+meaning (Floridi 2005: pp. 351–359). According to this analysis, an object O is an instance of semantic information if:

1. O consists of N data (for positive integer N), i.e. information cannot exist without data. For instance, a database search query that returns a negative answer, such as 'no entries found', yields some other positive data through either explicit negative data or some metadata about the search query.
2. The data are well formed. Semantic information depends on the occurrence of both syntactically well-formed patterns of data and physically implementable differences.
3. The data are meaningful[6]. The question here is *whether* data conveying semantic information can be rightly described as being meaningful independently of the recipient.

The resulting account of semantic information clearly depends on the particular interpretation of *data*. As noted above, the standard way to interpret data is as physical symbols used to represent information. But this results in a (nonvicious) circular definition. In response, Luciano Floridi proposes a somewhat different definition of data that does not explicitly refer to information (2011: pp. 85–86). Datum *d* is defined in terms of uninterpreted variables that are distinct from one another in a domain that is left open to further interpretation. The actual format, medium and language in which semantic information is encoded may remain unspecified in the general case. Information may be expressed verbally, pictorially and using arbitrarily different languages.

Yet, the standard analysis of semantic information is arguably insufficient, as it does not require that semantic information be veridical. Meaningful and well-formed data still qualify as semantic information, regardless of whether they represent or convey a truth. Thus, on the standard analysis, misinformation, disinformation and tautologies all count as genuine types of semantic information (Floridi 2005: pp. 359–360; Dretske 1981: pp. 44-45). According to the veridicality thesis, semantic information must be truthful. It has to represent true contents about the referred object, event, topic or state of affairs. So, the standard analysis has to be modified to also include the requirement that *the data be truthful* (Floridi 2005: pp. 365–366).

On the one hand, the first type of semantic information, *factual* information, represents facts or states of affairs and only qualifies as information if it is true.

[5] UTMs differ in implementation resulting in the informative content of a string being relative to the particular UTM used to calculate its AI complexity, K. Cristian Calude shows that for every two UTMs u_1 and u_2 $\forall x \exists c$: $(x \in S, c \in N)\ |Ku_1(x) - Ku_2(x)| \leq c$ where x is the input to the UTM (and S is the set of all strings) (2002: p. 38).

[6] This principle tacitly assumes the existence (even in the past) of some agent with a system of values relative to whom the data are (or were) meaningful.

Some philosophers (including Dretske 1981, Barwise and Seligman 1997, Floridi 2009, 2011) have argued for the alethic (i.e. truth-based) nature of factual information, whereas others (including Carnap and Bar-Hillel 1952, Fetzer 2004, Scarantino and Piccinini 2010) have argued that it need not be truthful. Roughly speaking, those who argue for the alethic nature of factual information, also assert that it is only *true* factual information that yields knowledge and reject misinformation and disinformation as genuine types of information. And those arguing against the veridicality of factual information also agree that meaningful and well-formed data already qualify as information, regardless of being true or false.

The debate about whether semantic information indeed must be truthful remains unsettled. Some philosophers (most notably, Rudolf Carnap and Yehoshua Bar-Hillel) argued against the veridicality of semantic information. Carnap and Bar-Hillel (1952) characterise the semantic information in a sentence (or proposition) S in terms of a logical probability space and the inverse relation between information and the probability of S. It is generally accepted that tautologies convey 0 information for they have an "absolute logical probability" (ibid: p.2) of 1 (Floridi 2011: p. 99). However, when considering contradictions, whose probability is 0, they supposedly contain the maximal semantic information content for the very same reason (Carnap and Bar-Hillel 1952: pp. 7–8; Floridi 2009: p. 44).[7] According to Carnap and Bar-Hillel (1952: p. 8), this consequence is not problematic, since on their view, semantic information does not imply truth. A false proposition could be highly informative in their pragmatic sense. However, this claim is highly contestable and leads to some paradoxical results.

On the other hand, *instructional* information (the second type of semantic information) seems to be a more appropriate candidate for an IP account of computation. The first three standard principles of semantic information are sufficient for instructional information. This information is not about some fact or state of affairs, so it cannot be correctly qualified as true or false. Still, it can be related to descriptive information, such as declaring a variable of a certain type in computer programming. It is meant to help produce some state of affairs. An instruction manual for a washing machine contains instructional information (either imperatively or conditionally) to help produce the expected result of clean clothes. This information is not conveyed factually, but rather either imperatively (step 1, do this; step 2, do that, etc.) or conditionally (if X do this, otherwise do that). The operation of logic gates of a computer, for instance, can be described in terms of conditional logic instructions for channelling the gates' electric voltages.

3 The Resulting IP Accounts Based on the Different Interpretations of Information

An SI-based IP account is, at best, limited in its ability to explain discrete deterministic computation in physical systems. Such an account clearly has some merit for explaining concrete computation. SIT emphasises the role of symbol structures as designating a particular state of affairs out of some larger set of possible states (i.e. selective

[7] This counterintuitive consequence is known as the Bar-Hillel-Carnap paradox (Floridi 2011: p. 100).

information; Ralston 1976: p. 647). SI is fundamentally a non-actualist conception of information, for it considers the actual world as simply one of many other possible worlds.[8] Accordingly, the actual message transmitted is just one of many other possible messages. To a great degree, conventional digital computers are non-actualists, in the sense that they are designed to resist minor perturbations (e.g. noise) and respond to a broad class of inputs. The non-actualist character of digital computing systems suggests some compatibility between them and SI.

Indeed, the selective characteristic of SI is compatible with a particular control structure allowing programmable computing systems remarkable flexibility. Selective information is closely connected with the way in which digital computers are capable of performing a conditional branch. This operation detects which of several different states obtains (e.g. what input was entered or which symbol was last scanned by the TM) and then sends the computation along different paths accordingly. The use of selective information by conditional branch processes lies at the heart of everything complex that digital computers do (ibid).

Furthermore, hardware malfunctions in deterministic computing systems could be described as noise in discrete communication channels. SIT deals with those aspects of communication where a transmitted signal is perturbed by noise during transmission. The received signal is then analysed as a function of the transmitted signal and a probabilistic noise variable (Shannon 1948: pp. 20–25). When considering a miscomputation, which is the result of some hardware malfunction, as a malformed signal, an analysis of it in terms of SI processing can be useful.[9] The computer's memory registers, for instance, are designed to handle such noise by including error correction mechanisms using parity bits, information redundancy etc. An even more obvious example is a discrete neural net one of whose units has lost the ability to transmit and/or receive signals to/from some neighbouring units. Such a scenario can be analysed in terms of noise and the channel's capacity to send and receive messages.

Yet, SI is a probabilistic concept whereas digital computation may be either deterministic or not deterministic (e.g. probabilistic computation, pseudo-random computation, etc.). The description of a physical system in terms of SI is only adequate if it is memoryless (i.e. if the system's transition to state S_{j+1} is unrelated to its last state S_j). But this is typically not the case for most conventional digital computing systems, which are deterministic, for their behaviour is repeatable and systematic. A dry run of a deterministic algorithm (using some test data) systematically yields the same output when its input and initial state remain unchanged. Similarly, the activity across the intermediate layers in a feedforward neural net is determined by the connection weights amongst units and the units' threshold functions.

[8] This is consistent with the possibilist's thesis (in the metaphysics of modality) that the set of all *actual* things is only a subset *all* of the things that *are* (possible).

[9] To a first approximation, a miscomputation is a mistake in the computation process due to a hardware malfunction or a runtime error of the executed program. A runtime error is typically the result of mistakes made by the programmers or designers of the program producing an incorrect or unexpected behaviour at runtime. Less common are errors that are caused by compilers producing an incorrect code (but even those can be attributed to human errors in the complier program). Common examples of runtime errors include the program running out of available memory, attempting to divide by 0, accessing illegal memory locations (e.g., when attempting to read past the last cell of a data array), or dereferencing a NULL pointer, which no longer points to a valid memory location.

Still, it is important to note that to some extent, when subject to noise even so-called physical deterministic computing systems are actually *not deterministic*. For there is an element of uncertainty introduced by a possible malfunction of any of the system's components. Most modern digital computers (e.g. particularly those based on multiprocessor or multicore architectures) regularly perform multitasking (leading to interaction amongst processes or amongst threads within processes). And this makes it very difficult to regard the execution of any single program strictly as a conventional single-tape, single-head TM, for a single program may include multiple threads blurring the classical division of computational labour amongst sequential procedures. As well, the multiprocessing (or multithreading) approach is susceptible to performance problems due to unpredictable delays whilst processes (or threads) are within critical sections.

Moreover, these possible delays may make the results of the overall computation unpredictable. Some of these unpredictable delays could be the result of caching or paged memory management, which are under the control of the computer's operating system (hereafter, OS) with support from the hardware. These delays could also be the result of the "external" environment, such as external devices (e.g. a mouse, keyboard, USB flash drive, etc.) or communication lines to other computing systems (e.g. through modems, old serial ports or Bluetooth enabled devices). And even low-level programming languages (most notably, assembler), which have some access to the hardware level, cannot deal with such (external) noise (through error recovery mechanisms) (White 2011: pp. 192–195). Besides, even in the absence of external noise, it is impractical to enumerate all the possible paths through a large nontrivial program (Gruenberger 1976: p. 189). This suggests that deterministic computer programs too are only deterministic to some degree of idealisation.

Whilst the state-transitions of Shannon's communication model are probabilistic, the transition probabilities of a *deterministic* TM are all set to 1. For every possible input (and a given initial state), there is only one possible state into which the TM transitions (Broderick 2004: p. 8). As Alan Turing stated "given the initial state of the machine and the input signals it is always possible to predict all future states" (1950: p. 440). Every future state transition can be accurately predicted by simulating the program being executed. So, in the case of idealised TMs, there is no element of uncertainty or surprise, on which SI is based. But in the case of conventional digital computers, which are susceptible to noise (at both the software and the hardware levels), an SI-based IP account may be useful in describing such (potential) adverse effects on (otherwise) deterministic computations.

Nevertheless, an SI-based IP account can only tell us what the lower bounds are for any solution to a given computational problem. SIT provides mathematical measures to calculate the *lower* bounds of information flow along a channel. It can tell us that a solution to a given problem cannot be computed in less than n bits of information. But an SI-based analysis cannot distinguish between two equally small circuits or different optimal programs that solve the same problem (and there are infinitely many such programs).

AIT, on the other hand, can distinguish amongst different optimal programs that solve a specific problem. In principle, the set of all such possible optimal programs is enumerable (Calude 2009: p. 82). And the full description of each one of these enumerated programs could be provided. However, traditional AIT can only *approximate* the

complexity of a string relative to a particular UTM and this prevents us from actually having a full description of all the optimal programs producing that string (Calude et al. 2011: pp. 5668–5669). Still, a variation on traditional AIT, which is not based on UTMs, allows us to compute the complexity of strings (or the exact length of these optimal programs), but this comes at a cost.

Finite-state transducer AIT (henceforth, FSTAIT) relies on finite transducers for which the universality theorem[10] is false. A transducer, in this context, is a finite state automaton with outputs. It can be described as the six-tuple (Q, Σ, Γ, q_0, Q_F, E), where Q is the finite set of states, Σ and Γ are the finite sets of input and output symbols, respectively, $q_0 \in Q$ is the initial state, $Q_F \subseteq Q$ is the set of accepting states and E is the finite set of transitions (ibid). Since there is no universal transducer, FSTAIT cannot achieve universality.[11] At the same time, the finite state complexity of a string explicitly includes the size of the particular transducer running the program and this complexity becomes computable (whereas traditional AI complexity can only be approximated). For our purposes, this means not only that different optimal programs are enumerable, but also that they are *distinguishable* from one another.

Furthermore, some aspects of the implementing machine are also taken into account by AIT. The machine's size is included as part of the encoded length of the computed string (ibid). When AIT examines which problems can be solved, a *particular* self-delimiting UTM is considered (or a specific combination of finite transducers for FSTAIT). This UTM could be based, for example, on a register machine with a finite number of registers (Calude 2009: pp. 82–83). The possible instructions of the program would also be considered, such as:

1. EQ R1 R2 R3 (if-then-else conditional instruction)
2. SET R1 R2
3. ADD R1 R2
4. READ R1
5. HALT

In the light of the above considerations, an AI-based IP account seems to do better than an SI-based account in explaining some physical aspects of computing systems as well as the set of instructions driving the computation. Moreover, in a manner similar to SI, AI complexity is a non-actualist conception of information. In accordance with the universality theorem, any program in some computer language can be converted (or compiled) into a program running on a UTM. There is some algorithmic procedure for passing amongst the possible enumerated TMs that compute a function f. To that end, we can pretend that we have all these enumerated TMs in front of us. If, for instance, we need to compute 20 steps in the computation of the fifth

[10] This theorem states that there exists a self-delimiting UTM U, such that for every self-delimiting TM T, a constant c can be computed (depending only on U and T), satisfying the following property. If T(x) halts, then $U(x')=T(x)$, for some string x' whose length is no longer than the length of the string x plus c (Calude 2009: p. 81).

[11] There is no finite generalised transducer that can simulate a transducer running some program. Yet, Calude et al. (2011: p. 5672) prove that the invariance theorem (informally saying that a UTM provides an optimal means of description up to an additive constant) also holds true for finite state complexity. Finite state complexity of a finite string x is defined in terms of a finite transducer T and a finite string s such that T on input s outputs x. It is defined relative to the number of states of transducers used for minimal encodings of arbitrary strings.

machine on input string "slam dunk", then we simply pick the fifth machine, put "slam dunk" on its tape and run it for 20 steps (Downey and Hirschfeldt 2010: p. 9).

In order to assign AI complexity to the configuration of some computing system (both the machine and the self-delimiting program it simulates), the system is "frozen" at some point in time. That snapshot of the *actual* computation taking place (rather than all possible counterfactual computations) is assigned an AI complexity value. Still, since all the enumerated TMs for computing *f* are available in principle, the AI complexity analysis is not strictly limited just to the actual computation that is executed.

As for the resulting IP account that is based on *semantic information*, its basis could be either *factual* or *instructional* information. If we take semantic information to be factual, then any processing of information is both *representational* and *truth preserving*. Symbolic representations (or subsymbolic representations in neural nets), which are processed by the computing system, supposedly carry true information about some (possibly external) state of affairs. The preservation of the isomorphic mapping between these representations and the state of affairs in question requires that their processing be truth preserving. Unlike SI and AI, factual information is a thoroughly *actualist* conception. It refers to situations that either have been actualised or are actualising in the world. A factual assertion commits the asserter not to its truth in some possible world but in the actual world. It seems then that processing of factual information and concrete computation are incompatible, for the latter is non-actualist.

This incompatibility raises a problem in regard to concrete computation in human-engineered systems. A digital computing system allows all the *possible* computations, which the system *could have* performed, rather than just the one it *actually* performs (this point is used to block Searle-like trivialisation of computation and is discussed at some length in Fresco (2011)). So if we opt to explain the working of the computing system in terms of information processing, then the explanans should be given by the relation between that system and the world in all possible worlds in which the computation may actualise. Still, an IP account based on factual information can only explain one particular scenario of computation, namely the one actualising in the real world.

On the other hand, an IP account based on instructional information seems less problematic. The processed information is not characterised as true or false. A program executed on a conventional digital computer could be interpreted as the execution of instructional information (e.g. assign the value '3' to variable var1, do *X* if *Y*, otherwise halt; where *X* stands for an instruction and *Y* is the condition). Similarly, the working of a neural net can be explained in terms of instructional information. The local rules of the individual units' threshold functions are explicable by conditional information (e.g. if the summed input is greater than 0, then fire, otherwise remain inactive). The interconnection weights are explicable by imperative information (e.g. assign the value "2" to the connection between units X_1 and Y_2).

Furthermore, the underlying hardware of the computing system could be described as the flow of electricity through its logic gates yielding its discrete state-transitions (when they stabilise). Computer programs and hardware are based on algebraic rules in which 'true' and 'false' simply correspond to 1 and 0, respectively. Conspicuously, logic gates are implemented as electronic switches whose entire operation is based on

manipulating electrical voltages representing the binary states 0 and 1 (e.g. 0 and 5 V representing 0 and 1, respectively).

Additionally, instructional information is not limited to describing actual computations only. The informational content of a program is given by its behaviour on all possible inputs, rather than just on actual inputs. UTMs (and similarly conventional digital computers) may take the same vehicles (be that strings, numerals or anything else) as either instructions or data that are operated upon according to some instructions. The very same string may play the role of an instruction in one run of the program and data in another (Scarantino and Piccinini 2010: p. 326). The same string may even be both an instruction and data during a single run of a program. Instructional information is certainly compatible with this principle. There is nothing in the definition of instructional information that implies that it only applies to actual occurrences, for it is not evaluated alethically.

In sum, the four different interpretations of information give rise to different IP accounts of computation each of which can explain certain aspects of concrete computation. Section four specifically examines the particular problems that each resulting account faces. But before we can settle on a specific IP account as being adequate, the next section examines the key requirements implied by each resulting IP account.

3.1 The Key Requirements Implied by the Resulting IP Account

The key requirements for a physical system to perform nontrivial digital computation[12] implied by any IP account are fourfold: (1) having the capacity to send information, (2) having the capacity to receive information, (3) having the capacity to store and retrieve information, and (4) having the capacity to process information. The fourth key requirement is affected by the particular interpretation of information (as is shown below). Importantly, whilst not strictly implied by the four interpretations of information, a fifth requirement is needed to exclude systems that only perform trivial computations. This requirement is having the capacity to actualise control information.[13] For simplicity, the following discussion remains neutral on the specific interpretation of information, unless specified otherwise.

The first key requirement implied by any resulting IP account is the system having the capacity to send information. The sender prepares the messages to be sent to the receiver and encodes them for transmission. An important distinction that should be drawn in the context of digital computing systems is between sending information *internally* amongst different components of the same system and *externally* between the system and some external interface (e.g. an input/output device or another computing system). A computing system devoid of any external interfaces may still

[12] One might question the reasoning behind the key requirements coming from the resulting IP account, rather than coming from actual computing systems. Once these key requirements are explicated, then various interpretations of information can be evaluated as a basis for an IP account. But that would be missing the point, for it is not at all clear what it takes for a physical system to compute. This is also the reason for the existence of many extensionally different accounts of computation (Fresco 2011). The IP account is only one of them, and it is commonly invoked in cognitive science.

[13] At the same time, this requirement makes it less obvious how discrete connectionist networks perform digital computation in the absence of explicit control units. This complication shall not be further considered here.

compute a solution to some predefined problem. An example of sending information internally is the computer's main memory being the *source* of information (e.g. a stored instruction). The memory controller acting as the *sender* is responsible for fetching data from the main memory and transmitting them to the CPU. Similarly, the tape of a TM can be regarded as the source of information when its head (acting as the "sender") reads a symbol from the tape.

Moreover, at times, a digital computing system acts both as a sender and a receiver. One example is the units of a discrete neural net. Any unit in the input layer receives information from another system (or possibly through some feedback loop in the net) and sends information to one or more units in the subsequent level (and conversely for the output units). Any unit in the intermediate layers receives information from at least one other unit in the preceding layer and sends information (assuming its threshold function was met) to the units connected to it at the next level. Similarly, a transmission of information internally can take place between two programs running on a computer or between the computer's memory (the source) and the CPU (the destination).

Analogously, the second key requirement implied by any resulting IP account is the system having the capacity to receive information. But in the context of computing systems, this requirement needs to be relaxed a bit. If the former requirement necessitated a sender to transmit the message, this requirement expects a receiver on the other end to accept it, at least in principle. In some cases, the absence of a receiver on the other end means that the computation remains unexecuted (or in a suspended mode). For instance, a program thread (the sender), which sends an input/output signal to the OS (acting as the receiver), will enter the suspended mode until its I/O request is acknowledged. But whereas a sender is needed to transmit the information, there are cases where the absence of a receiver does not result in an incomplete computation. This is particularly common in intercomputer communication, but applies to individual computing systems just the same.

One example is certain communication protocols invoked amongst various computing systems and another is some units of a neural net acting as senders without receivers. Some communication protocols such as TCP, which is at the heart of all HTTP-based Internet transactions, require a "handshake" between the sender and the receiver for the transaction to be successful (e.g. consider the "Server not found" error message displayed when a particular website cannot be reached). But other protocols such as user datagram protocol (UDP) do not require that the receiver acknowledge the receipt of the message sent (and are less reliable, but faster). Moreover, in feedforward networks it is possible that some particular units will act as receivers (or senders) but not as senders (or receivers) when the connections to other units become inactive. Although this may affect some local operations, the overall operation of the neural net as a whole may still be completed successfully.

Also, any resulting IP account typically requires a sender and a receiver that are well coordinated. Unlike a microphone acting as a sender of information even in the absence of a receiver (the audience), in computing systems senders and receivers are typically well coordinated. The information contained in a message may indeed not depend on the receiver's learning something from it, or even being able to decode it (Dretske 1981: p. 57). But if the receiver is unable to decode the message in a computing system, the computation will be either incomplete or incorrect (when considering cases where both the sender and receiver are *internal* to the computing system).

Suppose that the CPU (the receiver) does not correctly decode the instruction from the main control unit (the sender). This could be the result of noise on the channel (e.g. a hardware malfunction of some sort) or lack of synchronisation between the main control unit and the CPU. And if the error is not corrected, the execution of the instruction will fail, thus hindering the overall computation. However, in other cases, such as when using the UDP protocol discussed above, senders and receivers are not necessarily well coordinated. Whilst this option may be useful for time-sensitive programs, it is unlikely to be an option for real-time systems requiring high-reliability of the data transferred.

The third key requirement implied by any resulting IP account is the system having the capacity to store and retrieve information.[14] The storage and retrieval of information in a computing system are well synchronised, as one always presupposes the other. Without the system having the ability to retrieve data, there is clearly very little sense to storing data in the first place. For instance, if the OS were stored in the computer's random access memory (RAM) instead of on the hard-drive or some other persistent memory medium (where it was expected), the computer would fail to start following a reboot.[15] The computer's RAM is a volatile memory, which retains data only as long as it has a power supply available. So that the computer can load its OS, the OS has to be stored on some persistent memory medium (e.g. hard-drive, read-only memory or flash memory). Otherwise, the result will not be a miscomputation, but rather the absence of any computation.[16]

Lastly, the fourth key requirement implied by the IP account is the system having the capacity to process information. This requirement is the essence of information *processing*. It is also the most problematic one and it becomes even more stringent when information is interpreted as *factual* information. It is important to emphasise that processing information does not merely amount to encoding and decoding information. Those are methods that typically preserve the information whilst converting it into a coded form and vice versa. *Processing* of information should not be simply equated with *transformation* of information either. Whilst transformation may be the modification or removal of (some) information, it does not imply the production of *new* information. Transformation (of information) implies a prior form (of information) changing to another form.

To avoid ambiguity, processing of information is characterised here as the production of new information, modification of information or its removal.[17] The production of new information, such as a new database table containing salaries of

[14] This requirement is problematic for most neural networks, for they typically lack the flexibility enabled by long-term memory. That is particularly problematic for nets lacking any feedback loops.

[15] It does not follow though that some working OS threads (or even the OS in its entirety) cannot be loaded onto RAM once the OS has finished loading from the persistent memory.

[16] In conventional general-purpose computers, the OS is the core program required for any other program to run. But if neither the OS (as the main executed program) nor any other program (by implication) is running on the computer, then effectively no computation is taking place.

[17] Information processing may be construed in a variety of ways depending on the particular context of enquiry, including (but not limited to) the manipulation, acquisition, parsing, derivation, storing, comparison and analysis of information. However, it seems to me that these depend crucially on at least one of the aforementioned operations. Also, insofar as processing of information is taken as a physical process, in accordance with the second law of thermodynamics, it always results in some change in free energy (Karnani et al. 2009). Thus, even when certain information is deleted from a computing system, it is not completely destroyed, for some energy dissipates from the system into its surrounding.

employees,[18] may be the derivation of new propositions from existing ones. The modification of existing information, such as giving some employees a pay rise, is the manipulation of some information I_1 such that $I_1 \neq I_2$ (where, I_2 is I_1 that is modified at T_{n+1}). The removal of information, such as deleting from the system matching records of employees, who left the company, is a selective removal of information that need not result in the deletion of the entire system.

Additionally, construing deterministic computation as information processing requires more than the communication of information in a nondeterministic manner. Computers encode, decode and transmit information (and so do telephones), but they also perform tasks with inferential import (when trying to divide a number by 0, a good program should yield an error message from the computing system). This requires a way of distinguishing the differences between the informational *contents* of the messages (i.e. the specific information these messages carry or their semantic content). SIT provides the procedures for selecting messages, but not to distinguish between their informational contents (Dretske 1981: p. 6).

This ability to distinguish between different contents is necessary for modifying or adding new justified information. SIT tells us about the probabilities associated with symbols from a given language, but it is indifferent to the content of the messages. For instance, the following strings S_1 and S_2 have the same length (including that of their symbol constituents); S_1="all cars have four wheels", S_2="all cats have four ankles". Let us suppose that S_1 and S_2 are equiprobable (so according to SIT, they are potentially equally informative). Let S_3 be "bumblebee is a car". By using universal instantiation and modus ponens (taking these as being represented in first-order predicate logic), one can infer some new justified information.[19]

For example, one can infer S_4="bumblebee has four wheels" from S_1 and S_3. The overall informative content of any two (different) strings combined is typically greater than that of each one of these strings individually.[20] This new information must also be true, if S_1 and S_3 are true. It tells us something else about bumblebee (namely that bumblebee has four wheels). S_2 and S_3, however, do not yield new justified information using universal instantiation (and modus ponens, similar to S_4). One cannot validly infer any new singular statement about bumblebee from the universal statement S_2. In order to apply rules of logic as a means of producing new true information, the symbolic constituents of strings must be distinguishable.

[18] Yet, it remains to be seen whether this new information stored in the database of a digital computing system is merely a *copy* of the (new) information that was created externally (e.g., by the human resources manager).

[19] There is an ongoing debate regarding information in deductive inferences. Some, including John S. Mill and the logical positivists, have argued that logical truths are tautologies, and so deductive reasoning does not add any *new* information. On this view, all valid deductive arguments simply beg the question. Others (notably, Jaakko Hintikka (1984)) have argued that deductive reasoning can indeed produce new nontrivial information.

[20] Generally, the amount of information in any two strings Si and Sj is not less than the sum of the information of Si and Sj, if the content of Si and the content of Sj are in some sense independent (or at least one does not contain the other). Still, there are clearly cases where $INF(Si+Sj) < INF(Si)+INF(Sj)$. For a more detailed discussion of the "additivity" principle see Carnap and Bar-Hillel (1952: pp. 12–13). Arguably, universal instantiation and modus ponens, for instance, as a means of inferring S_4 from S_1 and S_3 also carry some positive information, since without the recipient knowing how to use them, she cannot infer S_4.

But according to SIT, we may encode and transmit S_2 (rather than S_1) and S_3 to the recipient (as S_1 and S_2 are equiprobable) that learns nothing new from S_2 and S_3 in this case.

Furthermore, when information is construed as *factual* information its processing requirement becomes even more stringent. The syntactical manipulation of messages must be done in a manner that preserves their semantics. Typically, rules that are applied in the processing operation must be truth preserving.[21] At the very least, new justified information has to be consistent with prior existing factual information. If conjunction, for instance, is applied to produce new justified information, then the conjuncts C_1 and C_2 must be neither contradictories nor contraries. Otherwise, their conjunction $C_1 \wedge C_2$ would be false (thus, nonfactual).

A database-driven computing system that progressively produces new information is a good example. Consider a system, whose database is initially populated with some basic propositions, designed to progressively increase the overall information based on these initial propositions. If the information processed by the system is taken to be factual (and hence true), then the resulting new information must be true as well. The system could progressively produce more information by means of logical inferences. For instance, if propositions P and Q were entered initially, then the system could produce the new proposition $(P \wedge Q)$ and add it as a new entry in the database. Intuitively, the more propositions there are in the database, the more informative it becomes. But this requires that the system be capable of determining which propositions are true and which are false. Otherwise, inconsistencies and tautologies will eventually creep into the database, thereby decreasing its overall informative content.

Whilst the aforementioned requirements apply to many information processing systems, they are insufficient for excluding some systems that only perform trivial computations. A fifth requirement is thus needed to exclude those systems. Arguably, basic logic gates send, receive and transform[22] information. Flip-flops and shift registers can also store information. However, such systems perform only *trivial* digital computation. Let us grant that any component that takes one or more lines of input and produces a single output (of the same type), which stands in a definite logical relation to its input(s), may be described as a basic logic gate. Then logic gates can be built using water, instead of electricity, to activate the gating function. A hydraulic OR gate, for example, can be built by merging two water pipes. This gate *trivially* computes the logical OR function. Still, it seems excessive to classify such a gate as a digital computing system proper.

What basic logic gates, flip-flops and shift registers lack is a control unit. Put another way, in an information-theoretic language, digital computing systems proper have *the capacity to actualise control information*. Logic gates and flip-flops can be wired together to form computing systems, whose computations can be logically

[21] Induction, abduction and nonmonotonic logic do not abide by the same principle, and their application does not guarantee the truth of any new information that they potentially produce. Both abductive reasoning and non-monotonic logic play an important role in artificial intelligence and should not be discounted, but they exceed the scope of this paper.

[22] The reader will have noticed that I have deliberately used "transform" here, rather than "process". For information processing (but not its transformation) also implies the (possible) production of new information.

analysed into the operations performed by their components. This is, after all, how finite state machines are built. Nevertheless, not every collection of entities, even provided that they may be described in isolation as logic gates, can be connected together to form a digital computing system. The inputs and outputs of logic gates are typically of the same type, so that outputs from one gate can be transmitted as inputs to other gates. The components must also be appropriately organised, synchronised (Piccinini 2007: pp. 522–523) and include some controller unit(s).

Specifically, control information in the context of digital computing systems is characterised as the capacity to control the acquisition and utilisation of information to produce a definitive action.[23] Accordingly, control information is distinguished functionally from the process of exercising control (Corning 2001: p. 1276) and it may only exist *in potentia* until the system actually uses it. Control information is always relational and context dependent and it does not exist independently of the computing process. Further, it has no fixed structure or value—a single binary bit may be sufficient for producing a definitive action (ibid: p. 1279), say, by expressing a *stop* or *continue* command.

Furthermore, control information is content-based and this affects its relation to the four interpretations of information. It is excluded from SI, since SIT ignores informational content. AIT, on the other hand, in grounding informational analysis on UTMs (or any other control mechanism) at the very least implicitly assumes control information. Still, it is instructional information that is most compatible with control information, for "a program is literally a description of what the controller [does] at run-time" (Larsson and Lüders 2004: p. 5).

Moreover, to ground 'control information' in the context of computing systems, let us consider the following cases. A numerical opcode in computer machine language represents the primitive operations supported by the particular computer architecture, say, an ADD operation. To execute this operation, the CPU follows the opcode direction to the physical address of the ADD operation. ADD is coded by a unique binary pattern and whenever this particular sequence lands in the CPU's instruction register, it is akin to a dialled telephone number that mechanically opens up the lines to the right special-purpose circuit (Dennett 1991: p. 214).

The controller receives the part of the instruction that encodes a command (such as addition, multiplication, storing a value in register R1, etc.). This command (i.e. the control information) is used by the controller to determine which operation needs to be performed on the corresponding data that are subsequently sent to another component for execution. Likewise (though in an idealised manner), a TM is controlled by a finite-state controller and at each step of the computation, the combination of the symbol read from the tape and the state of the TM determines its next operation. The control information of the TM is its m-configuration that specifies which instruction is to be followed next.

Control information is also used in network communication, whereas basic logic gates, shift registers and the likes are deprived of it. Control information is exchanged, for instance, between network routers for making routing decisions about sent messages. Further, to establish reliable communication lines and allow error recovery, control

[23] This characterisation is an adaptation of Peter Corning's teleonomic definition of control information in cybernetic and biological systems (2001: p. 1277).

information (e.g. unique network address identifiers) is sent and collected. However, whilst basic logic gates, flip-flops and shift registers may be senders, receivers and transformers of information, they lack the capacity to actualise control information even if they happen to transform it during their operation. Consequently, whilst digital computing systems proper exhibit the capacity to actualise control information, trivial computing systems (e.g. basic logic gates) do not.

4 Problems for the Resulting IP Account

4.1 Effective Computability as a Guiding Principle

Essentially, any account of concrete computation has to be able to explain at least the three key algorithmic notions of input, output and procedures. To begin with, any plausible account of computation has to be able to explain effective computability (whether in terms of TMs or not). Turing's analysis, for one, showed that instruction-following operations of a human computer could be simulated by TMs. His analysis identified effectively calculable functions with Turing-computable functions (Dershowitz and Gurevich 2008: p. 304).

Further, the subject matter of Turing's analysis is computability unlike the other formalisms of computability (Soare 2007: p. 706). His analysis studied what functions could be computed by a finite procedure. Whilst the other formalisms, such as Church's lambda calculus, Gödel's recursive functions and Post systems, are provably extensionally equivalent to Turing's, their subject matter is λ-definability, recursion and canonical systems, respectively, rather than computability directly. Turing was the first to provide a convincing definition of both a computable function and a UTM.

Also, in practice, Turing computability lays the ground rules for all existing digital computers as well as for all programming languages. Most modern programming languages (e.g. Ada, C, C++, Java, Lisp, Pascal etc.) are Turing complete, since (*if* equipped with unbounded memory) their syntax allows them to simulate any TM. The other formalisms (e.g. the lambda calculus) inevitably lead to a similar result too. All sequential functional programming languages, for instance, can be understood in terms of the lambda calculus, as it provides the basic mechanisms for the nesting of procedures. Incidentally, Alonzo Church stated that defining "effectiveness as computability by an arbitrary machine, subject to restrictions of finiteness, would seem an adequate representation of the ordinary notion" (as cited by Dershowitz and Gurevich 2008: p. 303). So any explanation of the operation of a digital computer or a computer program is underpinned by effective computability at some level of abstraction.

Moreover, the requirement to be able to explain effective computability also accords with some key criteria for evaluating adequate accounts of concrete computation.[24] For instance, Brian Cantwell Smith's (2002: p. 24) conceptual criterion and Piccinini's (2007: p. 504) explanation criterion mandate that any such account ought to explain underlying computational concepts (for example, a compiler, an interpreter, an

[24] For a detailed analysis of the key criteria for evaluating the adequacy of accounts of computation, see Fresco 2008.

algorithm, etc.) as well as how program execution relates to the general notion of digital computation.

Secondly, any account of effective computability has to explain at least the three key algorithmic notions of input, output and procedures. As Hao Wang put it "[w]hat is adequately explicated [by Turing computability] is the intuitive concept of mechanical procedures or algorithms or computation procedures of finite combinatorial procedures" (Wang 1974: p. 89). According to Turing (1936: pp. 231–232) a physical computing system has to recognise symbols, write symbols and store them in memory, change states and follow instructions. Following instructions amount to the computing system acting in accordance with an algorithm. The operation of a UTM is explained by its execution of the instructions of some special purpose TM on input that was also inscribed on the UTM's tape.

Even if we opted not to analyse concrete computation in terms of TMs, the fundamental notion of an algorithm would still require explication. Nachum Dershowitz and Yuri Gurevich, for instance, offer four postulates that allow a natural axiomatisation of computability (2008). Their analysis is not specific to any particular computational model, but it applies to arbitrary state-transition systems with arbitrary structures for states. It can also be generalised to encompass parallel interactive and parallel computations. At the heart of this analysis remains the underlying notion of an algorithm, which is characterised by the first postulate as determining "a sequence of computational states for each valid input" (ibid: p. 306). Therefore, any account of concrete computation has to *at least* explain the key notions of input, output and following procedures.

4.2 An IP Account Based on SI or AI

To start with, an IP account of concrete computation that is underpinned by SI has a limited explanatory power concerning effective computability. SI only makes sense in the context of a set of potential messages that are communicated between a sender and a receiver and a probability distribution over this set (Adriaans 2008: pp. 146–147). There is no room for a probabilistic selection of messages in describing deterministic procedures, for the probability is 1 (barring adverse effects of noise as discussed above). There must be a *specific set* of messages that are selected, encoded and transmitted in the same order in accordance with the specific steps of the procedure, regardless of the probabilities associated with each message (or its symbol constituents).

AI is an improvement on SI as a basis for an IP account of computation and does better in terms of explaining effective computability. AIT analyses the complexity of a string relative to a particular UTM as a single message and hides the probabilistic message selection process of SIT. AIT (and more specifically FSTAIT) can describe the behaviour of (optimal) programs (whilst industry computer programs are rarely ever optimal in the AIT sense). FSTAIT can describe the behaviour of non-optimal programs too, if the size of the program in bits is specified.[25] The resulting non-optimal programs

[25] Dealing with optimal programs is a feature of (at least conventional) AIT. But this by no means has any special bearing on AIT being an adequate candidate for an IP account of digital computation. Rather, the point is that AIT, unlike SIT, can adequately describe the behaviour of different programs.

then become enumerable and distinguishable from one another (as in the representative case of optimal programs; Calude, personal communication).

Moreover, as suggested above, the processing of SI or AI is problematic. The focus of SI is not on the content of individual messages, but that content *is* precisely what gets *manipulated*. Processing SI can be the modification of the state or string states that may result in changes of the conditional entropies amongst the states. It can also be the elimination of possibilities (reduction in uncertainty) represented by a signal or the introduction of redundancy to offset the impact of noise and equivocation. Still, sending the same message twice (as a means of introducing redundancy) does not yield more information than that in each.[26] Similarly, the elimination of redundancy does not reduce the underlying informational content. Importantly, whilst SI is viewed by many as having some potency as an instrument for creating order in complex systems, it is overrated (Corning 2001: p. 1274). Being insensitive to the content of individual messages renders the SI-based IP account incapable of distinguishing control information from other types of information.

Furthermore, noise on the channel is the source of modification and removal of information and uncertainty is the source of new information. Error correction methods are introduced as means of modifying information to offset that noise. But even then the underlying informational content of the messages remains (largely) unmodified.[27] And noise that causes the removal of (some) information is typically physical and rarely ever deliberate. We constantly try to find new ways to minimise the adverse impact of noise on communication (e.g. by using parity check bits, Hamming code, etc.). The deletion of information in computing systems, in contrast, could be completely deliberate, say, to free up memory resources or reduce the size of a database. As well, new SI is produced only relative to the uncertainty associated with that information. If the entropy of a message in a particular context is 0, then sending this message will not amount to producing new information.

In like manner, questions about the processing of AI amount to problems of encoding information. Producing new information, for example, amounts to the system producing an output string S_{OUTPUT} that *encodes* more information than the input string S_{INPUT}. For a computing system to be capable of producing new information it has to start with S_{INPUT} and produce S_{OUTPUT} with more information than S_{INPUT}. The production of new information amounts to the AI complexity of S_{OUTPUT} being greater than that of S_{INPUT}. Calude argues that a conventional digital computer can only produce limited new information upper bounded by a constant (2009: pp. 84–85). To show that this is the case, we need to find a self-delimiting TM, which halts on infinitely many inputs and is capable of producing infinitely many outputs each of which has more information than its corresponding input. But no TM is capable of such performance.

[26] It may be argued, however, that increasing the reliability of the message transmission process instils some confidence in the receiver. But even if that were the case, any "new" information here would remain constant and would not increase further by sending each message, say, three times (instead of two).

[27] Strictly, by adding, say, parity bits to a message M_1, the informational content in M_1 plus the parity bits increases over the informational content of just M_1. But unless those parity bits play an additional role as well as an error correction method (e.g., for data security as well as data integrity), the underlying information content is still conveyed by M_1.

Conversely, the deletion of information by a computing system amounts to the system starting with S_{INPUT} and producing S_{OUTPUT} with *less* information than S_{INPUT}. However, in this context it is worth reprising the distinction between data and information. A datum is defined in terms of uninterpreted variables that are distinct from one another in a domain that is left open to interpretation. So, a complete deletion of all data can only be achieved by the elimination of all differences amongst uninterpreted variables[28] (Floridi 2011: p. 85). A system that produces S_{OUTPUT} with *less* information than S_{INPUT} means that some unwanted information is displaced.

A simple example is a shift register that shifts in data entered as input bits and serially shifts out the last bit in the bit array. There are two modes of readout: destructive and nondestructive. In the former mode, the original data presented as input are removed once they have been shifted out of the right-most bit. However, in the latter mode, by adding an extra circuit, bits that are shifted out of the register are fed back into the system and hence not lost. Still, in the data transformation process, (some) information carried by the data is removed.

A similar principle holds for a selective deletion of certain entries in the database. Once the place in the memory holding that data (of those database entries) is overwritten, the (original) information is deleted. For example, the string "birtdayh happy" may be deleted from the database and be overwritten by "happy birthday". These are typical scenarios in classical computing systems. But they differ from the case of information dissemination within the computing system (e.g. when parts of the computer's memory are compressed and copied from one register to another) decreasing the system's descriptive complexity over time by means of self-organisation (i.e. by compressing and structuring unstructured information).

However, strictly speaking, an IP account based on AI will have a limited capacity to explain cases in which information is deleted and/or modified whilst the overall information complexity does not decrease. Or put another way, the system starts with an input string S_{INPUT} encoding less than or equal to the information encoded by the output string S_{OUTPUT}. Unless the particular UTM (or combination of finite state transducers) running the program is *changed* as a result of deletion of information, AIT cannot account for the deletion operation.

Additionally, the focus of AIT is the size of the program producing a certain output, but it ignores other practical considerations. For example, the "choice of computer or of computer programming language is not too important" for AIT (Chaitin 2007: p. 212). Chaitin's computer-as-a-decoder metaphor is useful to understanding *what* is computed, but not necessarily to *how* it is computed (when the AI complexity cannot be computed). FSTAIT allows us to also understand *how* it is computed for optimal programs (since AI complexity is computable).

It seems then that finite state transducer AI is a better candidate than classic AI for an IP account of computation, but both still face problems. The underlying physical architecture of the computing system and the supported instructions are also specified

[28] Floridi illustrates this point by considering a page of a book written in some unknown language (2011: p. 85). We have all the data but no information, for we do not know their meaning. If we erased half the content of that page, we might say that we have halved the data as well. Suppose we keep erasing the content of that page until the page is blank. Yet, we are left with *some* data, since the presence of the blank page is still a datum as long as it is different from a nonblank page.

and implicitly included in the calculation of AI complexity. Although the universality theorem does not hold for FSTAIT, the invariance theorem does. No physical computing system is a genuine implementation of a UTM (for Turing's unbounded memory requirement is violated), whereas computing systems can indeed be explained by the right combination of finite state transducers. Yet, both (conventional) AIT and FSTAIT deal with idealised computation. They deal with what happens between input and output whilst assuming *faultless* computation (Calude, personal communication). Any possible errors during the actual computation are ignored. AIT is based on idealised UTMs and (similarly) FSTAIT is based on faultless transducers.

4.3 An IP Account Based on Factual Information

Arguably, when information is interpreted as *factual* information it has to yield knowledge (Dretske 1981: pp. 45–47; Dunn 2008: p. 581; Floridi 2008: p. 118). That also implies a further requirement for the resulting IP account, namely that by processing information the computing system has to yield knowledge. This knowledge is either derived by its user (or programmer or interpreter) or intrinsic to the system. Plato defined knowledge as a true justified belief (which was widely accepted in modern philosophy[29]). Factual information must tell us something true about some state of affairs, that is, yield knowledge. One unproblematic option is that this knowledge is derivative and is used by the knower, who interprets the information produced by the computing system. Another option is that this knowledge is intrinsic to the computing system itself.

The latter option has been challenged by many philosophers (Agassi 1988, 2003: pp. 601–602; Dretske 1993; Dreyfus 1979; Harnad 1990; Penrose 1989: pp. 531–532; Searle 1980) and it is not at all clear that there is compelling evidence to support it. There is only a limited sense in which a digital computing system "understands" or "knows" anything. A digital computer only "understands" single-machine instructions (or multiple machine instructions simultaneously in the case of parallel computation) well enough to execute them. The CPU's "know-how" requires no (propositional) knowledge of what the primitive ADD operation is. The CPU just needs to follow the opcode direction of the ADD operation to its physical address and place some specified bits on the input lines of a logic circuit.

The semantics of these machine instructions can be traced back from the higher-level programming language (i.e. the particular problem solved by the executed program) through assembly language to the physical operation of the logic gates. The semantics of programming languages is formal and describes the relation between symbols in a computer language and their specific machine implementation. This formal semantics provides an abstract definition of the internal state of the computer and interprets the primitives of the programming language as actions on this state. A high level language, such as C++ or Java, describes the computer's state at a high level of abstraction referring to data structures and operations on them. A low-level language, such as C

[29] Edmund Gettier (1963) has challenged Plato's view of knowledge as Justified True Belief. He argued that truth, belief and justification are not sufficient conditions for knowledge. He showed that a true belief might be justified, but fail to be knowledge.

or assembler, describes labelled (relative) memory addresses and operations on them (White 2011: p. 194).

Moreover, at the program level, any factual information entered by the user is converted into something recognisable by the computing system by using an implicit *semantics dictionary*. This dictionary is used to translate factual information into some data structure that is recognisable by the program.[30] This program is then translated (either at runtime or at compilation time) into machine code that is executed (roughly speaking) on the machine hardware. At the hardware level, the working of the computing system is purely physical and is governed by laws of physics. Fundamentally, electrons flow through the system's logic gates. Still, whatever goes on at this level is completely determined by the programmed instructions and any input entered. The only semantics that exists at this level is internal to the computing system (e.g. instructions in certain memory addresses to be executed by the CPU, using certain registers for performing an addition operation and so on).

But it does not follow that the computer manifests any *beliefs* that are associated with these operations. Suppose we replace a doorbell with a digital computer that emits the sounds: "someone is at the door", only when someone pushes the door button. When someone pushes the button, the computer picks up the information about it, processes it and delivers an output. However, this output is not a belief that someone is at the door, anymore than the doorbell would have believed that (Dretske 1981: p. 204). Roy Sorensen (2007: pp. 158–179) distinguishes between information conveyed by assertions and displays. When a computer weather program displays a rainy weather forecast for tomorrow, it does not believe that it will rain tomorrow (though this output may be based on a reliable source of information). There is no relevant intrinsic belief or knowledge in these systems.

Moreover, digital computation will proceed (or not) regardless of the truth-value of the information processed by the computing system. Gricean non-natural meaning of signs (e.g. three dings of the bus bell indicating that the bus is full) does not require a correspondence to the state of affairs in question (e.g. whether the bus is actually full). Likewise, consider for example, a conventional computing system that was programmed with (or a neural net trained on) certain axioms and rules for inferring new propositions from old ones. This system may produce a particular output that corresponds to some state of affairs in world *X*.

However, if the same system was operating in another possible world *Y* (somewhat different from world *X*), it would still give the same output (assuming that the same input and that the initial state remains unchanged). But its output in world *Y* may not be *true* anymore. As far as deterministic computation goes, the future of the computational process is completely predictable from the very beginning in any possible world (save for some possible miscomputation).

Still, the information processed by the computing system need not correspond to an external state of affairs. Even if we took the input and the initial state as an external state of affairs (in the sense that they are set from outside the computing system), this would be the "point of departure" for the program execution. The program would

[30] The ace of hearts card, for instance, is represented as a data structure with properties such as a shape, a number etc. This data structure can be processed by the program and when appropriate, the processed data can be presented again in some form of human readable information as output.

proceed without necessarily preserving any correspondence to the relevant external state of affairs. Indeed, programs consist of well-formed data that are meaningful relative to the particular programming language. Therefore, they satisfy the principles of the common analysis of semantic information above and can be described as semantic information (sans truth value; Larsson and Lüders 2004: p. 5). The program can only be described as factual information *if* it happens to correspond to actual state of affairs. But whether a computation represents some state of affairs or not is a contingent fact.

5 A Plausible IP Account of Computation

Instructional information remains a plausible candidate for an IP account of computation. It is not susceptible to the problems that the other three interpretations of information face as candidates for an IP account. Instructional information cannot be characterised alethically as factual information. Also, it does better than SI and AI in terms of explaining computational procedures. Imperative (instructional) information is needed for explaining the default control structure that simply amounts to a sequential execution of instructions, such as finding the solution(s) of the algebraic equation $ax^2+bx+c=0$. Conditional information is needed for explaining conditional branching by allowing the program to follow alternative paths of execution by using *If X Then Y* or *If X Then Y Else Z* structures (where, *Y* and *Z* are instructions). It can also be used to explain looping over a certain operation (*Do Y While X)*. Combined together these two types of instructional information suffice to explain the operation of any Turing computable procedure.

Furthermore, an instructional information processing (IIP) account of computation is not limited to the program or algorithmic level. Conventional digital computers can also be explained in terms of IIP at the hardware level. But even the hardware level that is traditionally viewed as a single level can be further decomposed into several sublevels.[31] The following hierarchical decomposition shows how they differ and are underpinned by the lower levels.

- *Functional* level—the operation of the computing system is analysed in terms of the function being computed in the process of the underlying registers changing their stored values. At this level of abstraction, an IIP analysis is applicable in a similar way to the algorithmic level.
- *Register transfer* level (a technology-specific analysis)—the operation of the computing system is analysed in terms of the registers changing their stored values.
- *Logical* level—the operation of the computing system is analysed in terms of logic gates operating on various input lines. At this level of abstraction, it is easiest to see how these operations can be explained in terms of IIP of 0s and 1s.
- *Electrical* level (a technology-specific analysis)—the operation of the computing system is analysed in terms of electrical voltages, electro-mechanical operations, etc.
- *Physical* level—the operation of the computing system is analysed in terms of atomic or molecular movements.

[31] I owe this point to Karl-Christian Posch who suggested viewing these different levels of abstraction from an engineering perspective.

Viewed as a program-driven system,[32] an IIP analysis of a digital computer may be described as a continuous interaction amongst a *scanner*, an *interpreter* and an *operator*. Firstly, the scanner is responsible for reading an atomic instruction, encoding it as a message and sending the message to an interpreter. It can be thought of as the computing system's "head", which scans the lines of the program sequentially and transmits each atomic instruction as a message. Secondly, the interpreter receives a message, decodes and interprets it and then sends an encoded message (or messages) to the operator. There is no reason to assume a single interpreter, and indeed in some cases many interpreters could be at work.

Thirdly, the operator is the low level "worker" that fetches (and stores) information from (and in) short-term or long-term memory as well as sends commands for execution by the control unit (which actualises any control information). It is also likely that there would be multiple operators based on the various primitive operations, which are supported by the system's physical architecture. The scanner, interpreter and operator all play a dual role of both a source and destination depending on the particular operation in progress.

Consider the following pseudo-code procedure for multiplying two natural numbers.

```
procedure integer InefficientMutiply (integer multiplicand, integer multiplier){

integer multiplicationResult = 0;
if (multiplicand ≤ 0) return 0; // illegal input results in termination
while (multiplier > 0) do {

multiplicationResult = multiplicationResult + multiplicand;
multiplier = multiplier – 1;
}
return multiplicationResult; }
```

According to the proposed analysis above, the first few runtime iterations of this procedure will be processed in the following manner. The procedure is called with '3' as the multiplicand input argument and '4' as the multiplier input argument. The first three instructions are illustrated below (Figs. 1, 2, 3).[33]

Although the examples below (illustrated by Figs. 1, 2, 3) correspond to imperative, sequential programming, they apply equally well to other models too, such as the functional programming paradigm inspired by the lambda calculus. Whereas my examples illustrate computation in terms of sequences of instructions that change the program's (and machine's) state, the functional programming paradigm treats computation as a sequence of stateless function evaluation. One example of the functional programming paradigm is the PCF language used for game models. PCF is an applied, simply-typed lambda calculus that is built from a certain stock of constants, including first-order constants concerned with arithmetic manipulation and conditional branching (Abramsky et al 2000: p. 430).

[32] Consequently, a special purpose TM would require some modification of the proposed analysis, if we chose not to interpret it as executing a program, per se.

[33] Many intermediate steps have been removed for simplicity. For example, the name of the procedure "InefficientMultiply" in Fig. 1 is not added to the call stack (as it should be) allowing its retrieval at a later stage.

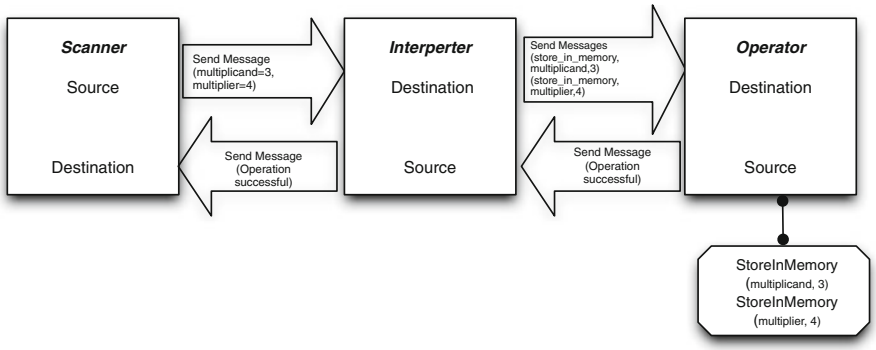

Fig. 1 First instruction call: invoking the procedure *InefficientMultiply* (*3*, *4*) using imperative information

More specifically, these game models are thought of as a series of moves between a player and an opponent (or a system and the environment) that may be finite or infinite. Each move by the player or its opponent is either a question or an answer. Questions are considered requests for input and answers are considered input or data (ibid: p. 414). The games are defined with global rules. These rules ensure, for instance, that computations evolve in a properly nested fashion. This exhibits a key structural feature of functional computation, what Abramsky et al refer to as the "switching condition" on pairs of successive moves (ibid: p. 417), that is essentially based on conditional branching.

Whilst functional programming (and computer game models) offers a different paradigm of digital computation, it can still be explained by the IIP analysis above. Such an analysis will have to explain the interactive approach of game models (being constantly driven by input requests and corresponding inputs) and the nesting of functions. But in principle, there is nothing crucially different in this paradigm that makes the IIP analysis inapplicable. The nesting of functions certainly exists in imperative programming as well (requiring additional communication between the interpreter and the operator).

Lastly, this proposed IIP analysis is suggestive and still requires some fine-tuning and several questions remain unanswered, to name just a few:

1. How does the scanner select the interpreter to use if the relation between them is not a one-to-one relationship, but rather a one-to-many relation?

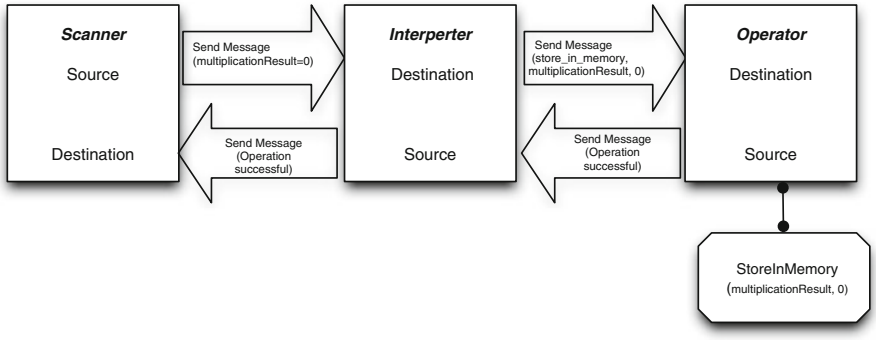

Fig. 2 Second instruction call: *multiplicationResult=0* using imperative information

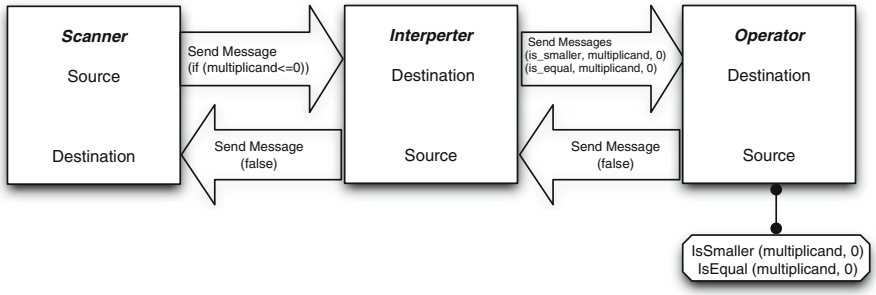

Fig. 3 Third instruction call: *if* (*multiplicand*≤*0*) using conditional information

2. This proposed analysis cuts across different levels of abstraction where the first two (scanner and interpreter) operate at the program level and the last one (the operator) operates at the machine level. Does it present a methodological problem for the IIP analysis?
3. How does the operator work at the relevant physical level discussed above?

6 The IIP Account Revisited

Importantly, whilst strictly SI-based or AI-based IP accounts are insufficient to explain concrete computation, they are systematically related to instructional information, which I argue is a plausible candidate for an IP account. SI is instrumental for the analysis of information flow in computer programs. For instance, when applying SIT to analyse security threats in computer programs, quantitative concepts such as 'entropy', 'channel capacity' and 'bit rates' become extremely useful. Pasquale Malacaria (2007) examines the maximum *amount of leakage* of loop constructs as a function of a potential attacker's knowledge of the program's input (on the basis of entropy) and the amount of information leaked as a function of the number of loop iterations (in terms of bit rates).

This program-security quantitative analysis exhibits a systematic relation between SI and instructional information. On the one hand, we measure how much confidential information is leaked to the potential attacker by the output of the program. The program is treated then as a black box and its output (that is visible to the attacker) is analysed in terms of SIT to quantify the information leakage. On the other hand, this analysis is based on the internal structure of the program, which leaks information. This internal structure is analysed in terms of instructional information (such as the system's states, relevant variables and loops or conditionals).

Moreover, AIT is inherently based on the *qualitative* notion of instructional information. AIT measures quantities of information in terms of the computational resources that are needed to specify it, viz., the program and the UTM (or finite state transducer, etc.) running it. This quantitative analysis is only possible when it is based on some basic set of instructions from which the program is composed. Otherwise, it would not be possible to measure the length of any optimal program for computing a particular string. AI complexity essentially depends on the number of instructions the program performs on a given input (if any), the computational capacity of the

particular programming language used (i.e. what basic instructions are supported) and the size of the program in characters (Calude 1988: p. 383).

Current AIT also requires that programs will be self-delimited so that their overall length can be calculated. And indeed programming languages are designed with that property in mind. There are always clear start and end points to every program and its procedures (e.g. in some languages a semicolon is used as an end-marker, in others the words 'BEGIN' and 'END' are used). Whether we choose to measure program-size in C, Fortran or LISP, the AI complexity analysis depends on the internal structure of the program in question. For instance, Calude uses a universal language that has seven basic instructions (plus input and output) and claims that this repertoire of instructions suffices for the creation of algorithms to solve all Turing-computable functions (1988: pp. 384, 402). These seven instructions are Assignment, Set to zero, Successor, Conditional, Loop, END and ';'. The internal structure of the program is plainly based on instructional information.

Finally, let us now examine several problems that were raised above in regard to an IP account of computation to determine their applicability to the IIP account.

- Is false information the same as a miscomputation?

Since, unlike factual information, instructional information is not characterised alethically, this problem can be sidestepped.

- What constitutes the production, modification and removal of instructional information?

Processing instructional information simply amounts to the execution of instructions plus (possibly) some input. If the computing system is instructed to populate the database with salaries of employees, it will do so whilst producing new information in the process. If a certain condition is met (say employees with more than 5 years seniority), the salaries of some employees will be updated (e.g. given a 10% a payrise) whilst modifying information (e.g. if E>5, then $S=S\times 1.1$). The removal of (some) information could simply be a deliberate deletion of some records from the database (when the employees have left the company), rather than being the result of noise that physically damages the database and deletes some records as a result (as in the case of processing SI, for instance).

- Does processing of instructional information have to be truth preserving?

Again, since, unlike factual information, instructional information is not characterised as true or false, processed information need not be truth preserving per se. Truth has to be preserved simply in a Boolean-algebraic sense. Say, if the condition (does X equal 0) is evaluated as true at T_1, then it should also be evaluated as true at T_2, assuming that the value of X has not changed between T_1 and T_2.

- Can the IIP account explain concrete computation as a function of the particular programming language used and the underlying physical architecture of the computing system?

Yes. Depending on the particular programming language, conditional information could give rise to different constructs of conditional branching such as loops, recursive procedure calls, goto statements etc. Imperative information is the basis

for the assignment of values to variables (e.g. X=5, Str='this is a string', INSERT INTO employees_table (first name, last name, role, department, start date) VALUES (Michael, Smith, Team_Leader, IT_Security, 1.1.2008), etc.). Further, the underlying physical architecture of the computing system affects the manner in which information is processed. The instruction (*if x+y equals 10*) would be evaluated differently at the machine level, if the architecture did not support the addition operation directly on the computer's memory banks. The information will be processed differently at the level of the register transfer, for instance. More transfer operations between registers would be needed to calculate ($x+y$) before the register value can be compared with *10*.

- Can the IIP account explain the key algorithmic notions of input, output and procedures?

Yes, instructional information is compatible with the key notion of procedures by definition. Procedures are sequences of instructions that are either imperative (e.g. assigning values to variables, accepting input, printing output, etc.) or conditional. These are exactly the two existing types of instructional information.

- Can the IIP account explain the digital computation performed by discrete neural nets?

This question presupposes that (non-simulated) discrete neural nets are genuinely digital computing systems. This is debatable and not immediately clear. But if they were indeed computational, the answer would crucially depend on whether the right way to explain their operation is algorithmic or not. This debate is unsettled.

7 Conclusion

Although an IP account, on the face of it, seems like a natural and promising candidate for explaining concrete digital computation, it is not as obvious as it first seems. Its explanatory power depends on what we take 'information' to be. I have argued that the capacity to actualise control information is essential for individuating digital computing systems proper. And whilst algorithmic information fares much better than Shannon information as a candidate for a plausible IP account of concrete computation, the resulting account still faces problems. The most adequate candidate for an IP account is instructional information, for it is compatible with control information and avoids most of the problems faced by other interpretations of information. An interesting question is whether discrete neural nets can be fully explained by such an IP account. But this remains to be seen.

Acknowledgements Part of this research was done during a visiting fellowship at the IAS-STS in Graz, Austria in 2011. Thanks to Gualtiero Piccinini, Oron Shagrir and Matt Johnson for useful comments on earlier drafts of this paper. I have greatly benefited from discussions with Naftali Tishby and Karl Posch on the mathematical theory of information and with Cristian Calude on algorithmic information theory and for that I am grateful. I would also like to express my gratitude to both Graham White and Marty Wolf, who refereed the paper and agreed to drop their anonymity in the process. Their insightful comments helped reshape and improve this paper significantly. I am indebted to Phillip Staines for his detailed comments and ongoing support. Earlier versions of this paper were presented at the 2010 AAPNZ conference in Hamilton, NZ, the 2011 AISB convention in York, UK and the IAS-STS fellowship colloquium in Graz, Austria. All

the people mentioned above contributed to the final draft of the paper, but I am responsible for any remaining mistakes. This paper is dedicated in loving memory of Moshe Bensal.

References

Abramsky, S., Jagadeesan, R., & Malacaria, P. (2000). Full abstraction for PCF. *Information and Computation, 163*, 409–470.

Adriaans, P. (2008). Learning and the cooperative computational universe. In P. Adriaans & J. van Benthem (Eds.), *Handbook of the philosophy of science, volume 8: philosophy of information, pp. 133–167*. New York: Elsevier.

Agassi, J. (1988). Winter 1988 Daedalus. *SIGArt newsletter, 105*, 15–22.

Agassi, J. (2003). Newell's list. Commentary on Anderson, J. & Lebiere, C.: The Newell test for a theory of cognition. *Behavioural and brain sciences, 26*, 601–602.

Barwise, J., & Seligman, J. (1997). *Information flow: the logic of distributed systems*. Cambridge: Cambridge University Press.

Broderick, P. B. (2004). On communication and computation. *Minds and Machines, 14*, 1–19.

Calude, C. S. (1988). Theories of computational complexity. Elsevier Science.

Calude, C. S. (2002). Information and randomness: an algorithmic perspective. 2nd edition. Springer.

Calude, C. S. (2009). Information: the algorithmic paradigm. In G. Sommaruga (Ed.), *Formal theories of information*. New York: Springer.

Calude, C. S., Salomaa K., Roblot, T. K. (2011). Finite state complexity. *Theoretical Computer Science, 412*, 5668–5677.

Carnap, R., & Bar-Hillel, Y. (1952). An outline of a theory of semantic information. MIT Research Laboratory of Electronics, Technical Report No. 247. Cambridge, MA: MIT.

Chaitin, G. J. (2003). Algorithmic information theory. 3rd Printing. Cambridge University Press.

Chaitin, G. J. (2007). *Thinking about Gödel & Turing: essays on complexity, 1970–2007*. Singapoer: World Scientific.

Cordeschi, R. (2004). Cybernetics. In L. Floridi (Ed.), *The Blackwell guide to philosophy of computing and information* (pp. 186–196). Oxford: Blackwell.

Corning, P. A. (2001). "Control information": the missing element in Norbert Wiener's cybernetic paradigm? *Kybernetes, 30*, 1272–1288.

Dennett, D. C. (1991). *Consciousness explained*. Boston, MA: Little, Brown.

Dershowitz, N., & Gurevich, Y. (2008). A natural axiomatization of computability and proof of Church's thesis. *The Bulletin of Symbolic Logic, 14*, 299–350.

Downey, R. G., & Hirschfeldt, D. R. (2010). *Algorithmic randomness and complexity*. NY: Springer.

Dretske, F. I. (1981). *Knowledge and the flow of information*. Cambridge, MA: The MIT Press.

Dretske, F. I. (1993). Can intelligence be artificial? *Philosophical studies, 71*, 201–216.

Dreyfus, H. L. (1979). *What computers can't do: the limits of artificial intelligence* (2nd ed.). NY: Harper & Row.

Dunn, M. (2008). Information in computer science. In P. Adriaans & J. van Benthem (Eds.), *Handbook of the philosophy of science, volume 8: philosophy of information, pp. 581-608*. New York: Elsevier.

Fetzer, J. H. (2004). Information: does it have to be true? *Minds and Machines, 14*, 223–229.

Feynman, R. P. (1996). *The Feynman lectures on computation*. Reading, MA: Addison-Wesley.

Floridi, L. (2005). *Is semantic information meaningful data?* (pp. 351–370). LXX: Philosophy and Phenomenological Research.

Floridi, L. (2008). Trends in the philosophy of information. In P. Adriaans & J. van Benthem (Eds.), *Handbook of the philosophy of science, volume 8: philosophy of information, pp. 113–131*. New York: Elsevier.

Floridi, L. (2009). Philosophical conceptions of information. In G. Sommaruga (Ed.), *Formal theories of information* (pp. 13–53). Berlin: Springer.

Floridi, L. (2011). *The philosophy of information*. Oxford, UK: Oxford University Press.

Fresco, N. (2008). An analysis of the criteria for evaluating adequate theories of computation. *Minds and Machines, 18*, 379–401.

Fresco, N. (2010). A computational account of connectionist networks. *Recent Patents on Computer Science, 3*, 20–27.

Fresco, N. (2011). Concrete digital computation: what does it take for a physical system to compute? *Journal of Logic, Language and Information, 20*, 513–537.

Gettier, E. L. (1963). Is justified true belief knowledge? *Analysis, 23*, 121–123.
Gruenberger, F. (1976). Bug. In A. Ralston (eds.) Encyclopedia of computer science. p. 189. NY: Van Nostrand Reinhold.
Harnad, S. (1990). The symbol grounding problem. *Physica, 42*, 335–346.
Hintikka, J. (1984). Some varieties of information. *Information processing and management, 20*, 175–181.
Karnani, M., Pääkkönen, K., & Annila, A. (2009). The physical character of information. *Proceedings of the Royal Society, A, 465*, 2155–2175. doi:10.1098/rspa.2009.0063.
Larsson, S., Lüders, F. (2004). On the concept of information in industrial control systems. In the proceedings of the National Course in Philosophy of Computer Science, pp. 1–6.
Malacaria, P. (2007). Assessing security threat of looping constructs. In the Proceedings of 34th ACM Symposium on Principles of Programming Languages, pp. 225–235.
Penrose, R. (1989). *The emperor's new mind*. London: Oxford University Press.
Piccinini, G. (2007). Computing mechanisms. *Philosophy of Science, 74*, 501–526.
Piccinini, G., & Scarantino, A. (2011). Information processing, computation, and cognition. *Journal of Biological Physics, 37*, 1–38.
Ralston, A. (1976). Encyclopedia of computer science. 1st edition. Petrocelli Books.
Scarantino, A., & Piccinini, G. (2010). Information without truth. *Metaphilosophy, 41*, 313–330.
Shannon, C. E. (1948). A mathematical theory of communication. *Mobile Computing and Communications Review, 5*, 1–55.
Smith, B. C. (2002). The foundations of computing. In M. Scheutz (Ed.), *Computationalism: new directions* (pp. 23–58). Cambridge, MA: The MIT Press.
Soare, R. (2007). Computability and incomputability. In S. B. Cooper, B. Löwe, & A. Sorbi (Eds.), *Proceedings of the third conference on Computability in Europe, Lecture Notes in Computer Science, 4497* (pp. 705–715). Berlin: Springer.
Sorensen, R. (2007). Can the dead speak? In S. Nuccentelli and G. Seay (Eds.) Themes from G. E. Moore: new essays in epistemology and ethics. New York: Oxford University Press.
Turing, A. M. (1950). *Computing machinery and intelligence* (pp. 433–460). LIX: Mind.
Wang, H. (1974). *From mathematics to philosophy*. New York: Humanities Press.
White, G. (2011). Descartes among the robots: computer science and the inner/outer distinction. *Minds and Machines, 21*, 179–202.
Wiener, N. (1948). *Cybernetics: or control and communication in the animal and the machine*. Cambridge: The MIT Press.
Wiener, N. (1966). God and Golem, Inc.: a comment on certain points where cybernetics impinges on religion. Cambridge: The MIT Press.