Noname manuscript No.
(will be inserted by the editor)

Algorithmic iteration for computational intelligence

Giuseppe Primiero

Received: date / Accepted: date

Abstract Machine awareness is a disputed research topic, in some circles considered a crucial step in realising Artificial General Intelligence. Understanding what that is, under which conditions such feature could arise and how it can be controlled is still a matter of speculation. A more concrete object of theoretical analysis is algorithmic iteration for computational intelligence, intended as the theoretical and practical ability of algorithms to design other algorithms for actions aimed at solving well-specified tasks. We know this ability is already shown by current AIs, and understanding its limits is an essential step in qualifying claims about machine awareness and Super-AI. We propose a formal translation of algorithmic iteration in a fragment of modal logic, formulate principles of transparency and faithfulness across human and machine intelligence, and consider the relevance to theoretical research on (Super)-AI as well as the practical import of our results.

1 Introduction

The notion of Artificial General Intelligence (AGI) is a hard one to pin down. Since the mid 1950s, several different and not always compatible definitions of artificial intelligence have been offered in the literature: from passing the Turing Test to showing common-sense knowledge in diverse situations. Each definition is often qualified in view of an appropriate method, discipline or objective. Today, the scientific research involved by this task spans over a combination of disciplines: logic, probability theory, machine learning, pattern recognition, vision, to name a few. In recent years, a number of academic and industrial areas have progressed in designing and programming machines that are significantly improving at *accomplishing tasks* that were previously the

G. Primiero
Department of Computer Science
Middlesex University
the Borroughs NW4 4BT
London
E-mail: G.Primiero@mdx.ac.uk

domain of humans, and doing so much better than us.[1] Nonetheless, the main principle of *true* AGI, intended as the creation of

> "[...] AI systems that possess a reasonable degree of self-understanding and autonomous self-control, and have the ability to solve a variety of complex problems in a variety of contexts, and to learn to solve new problems that they did not know about at the time of their creation" [2]

is a progressing but still far away objective, despite the always returning hype. Several different approaches can be subsumed under the task of obtaining AGI:[3] symbolic-processing, bio-computational, simulations, hybrid. For all of these, a reductionist definition assumes that resolved individual technical problems (e.g. in natural language processing, vision or path search) constitute the building blocks to be assembled in a complete solution, one whose external interpretation can be qualified as 'intelligent'. On this basis, let us start offering the following definition:

Definition 1 (Reductionist AGI) The epistemological and technological status of machines able to successfully perform any (composed) task feasible to humans.

This definition formulates intelligence in terms of observable behaviour and it does not specify any physical or otherwise qualified property that is essential to machines to perform the intended tasks. Nonetheless, this definition cannot be exhaustive. In particular, it does not account for the aspects related to self-understanding in terms of awareness and control of internal states highlighted in the quotation above and by many considered essential elements to qualify a truly intelligent mechanical behaviour. These aspects remain the most elusive ones: the machine not only needs to be able to manifest a behaviour that can be qualified as intelligent from the outside, it also needs to be aware of it.[4] Both believers in the possibility of Artificial (Super-)Intelligence and their detractors refer to consciousness as a cut-off point for *real* intelligence.

Believers in super-intelligence maintain that, given the current status of AI research and the foreseeable progress in both theory and applied technology, it cannot be excluded – and it is in fact likely – that the creation of a machine *"able to surpass all the intellectual activities of any man however clever"* will happen. As a consequence, *"an ultraintelligent machine could design even better machines"*, with the result of an *intelligence explosion*.[5] Since its original account, super-intelligence is formulated on the explicit condition that, in order to improve, the machine must have some understanding of itself. In recent, more technically nuanced analyses, the qualification of an artificial agent which attains super-intelligence through a series of self-improvements is identified with the ability to compare itself with previous systems equipped with progressively higher levels of intelligence, e.g. in terms of backwards induction (by computing its

[1] For a recent list of technical breakthroughs in AI see `http://futureoflife.org/2015/12/29/the-top-a-i-breakthroughs-of-2015/`. For a philosophical comment on how these represent more task solving successes than real progress in the mechanical definition of intelligence, see [27].

[2] See [37], p. VI.

[3] For a complete overview, including a historical presentation, see e.g. [58].

[4] See for example [72].

[5] Citations are from [38], considered the first speculation on machine super-intelligence. See [14] for the most up-to-date and complete analysis currently available for the problems, strategies and forecasts related to the possibility of an artificial super-intelligence.

actions in the future, and if so already at the initial stage) or by expected utility maximization [24].

Deniers of the super-intelligence trend similarly appeal to awareness, or consciousness, to debate the epistemological and technological credibility of any foreseeable true super-AI instantiation. A more technically challenging set of arguments against super-intelligence relies on computational complexity reasons to argue that the upper bound to human-level intelligence computable functions in the Arithmetical Hierarchy includes a class of problems whose complexity grows faster even than exponentially growing computable functions. It follows that there is no such (computable) function to express (and solve) these problems (not even by a super-intelligence) [74].

Awareness is therefore clearly crucial in the context of AGI. While the intuitive notion of intelligence undoubtedly appeals to a complex semantics of methods and abilities, a shift in focus can be offered by considering only knowledge states that can be already ascribed to *both* machines and humans. To refer to such kind of epistemic ability, we use the term *computational intelligence*:

Definition 2 (Computational Intelligence) The epistemological and technological status of entities able to successfully perform any (composed) algorithmically definable task.

This definition is based on the notion of algorithm and it is neutral with respect to the nature of the entities executing them.

The ontological and epistemological characterization of algorithms is crucial to Computer Science, and it has been object of recent formal and philosophical debates, [56,39,43]. Even from a purely technical perspective, the precise definition of the computational schema allowed by the notion of algorithm have been seeing a shift in understanding. Most notably, new forms of computation not strictly admitted by the historically qualified notion of mechanical computation by a Turing Machine reflect advancements in the required formal engineering: this is the case, for example, of interactive computing, where Turing Machines have been extended to work with infinite data streams [52]; reactive computing, where interaction in Turing Machines is modelled after concurrency theory [5]; distributed computing, where consistency, availability and partition-tolerance over data are exclusive properties [35,62]. To be able to offer a general argument on algorithms that does not exclude explicitly any of these forms, we will stick to the most abstract definition of the term: we intend an algorithm as a mechanically executable, ordered (possibly non-terminating) list of (multiple, interacting) steps for the application of computable functions on a well-defined domain of input values (be these simple or streams of infinite and complex nature). In this way, we try to capture the various forms of algorithms at work in modern computing, including e.g. interactive ones.

As we look at entities performing algorithmic actions, we are interested in the *implementation* of algorithms. Implementation in Computer Science is in itself a notion loaded with philosophical meaning, see [64,65]. The notion of computer program can be considered in at least four different ways:[6]

1. a technological object,
2. a mathematical object of finite capacity,
3. a mathematical object of infinite size, or

[6] See [21], Preface.

4. a model of the real world, which is not a logico-mathematical construction.

In what follows we will understand the term *program* as referring to the implementation of one such algorithm in a machine-readable language. By extension, a program can have a modular or distributed structure, including several algorithms.

Hence, on the one hand we look at algorithms as mathematical objects characterised by infinite size (i.e. without memory limitations); on the other hand, programs are considered implementations of algorithms *in any* language, executable *in principle*, i.e. mathematical objects of finite capacity. Fixing their characterization as mathematical objects, below we will make the explicit reference to program that can be proven formally correct with respect to their given specifications. In fact, we will restrict our analysis to specifications which can be proven to have a valid program. The actual execution of such a program, i.e. its actual transformation in syntactically correct constructs of a specific version of a given programming language and its actual running on a given hardware, refers to properties eluding the formal correctness we assume, and therefore they denote a different object than what considered in our analysis, namely a technological object. Accordingly, a machine is here intended as the abstract mathematical counterpart of a technological object. Design and execution of algorithms by a machine, exposed below, should be understood as appropriate mathematical idealizations of their concrete instantiations.[7]

Computational intelligence under the above definition and characterization of terms is clearly a much weaker object of analysis than GAI, but it has the advantage of offering a solid starting point. Any form of intelligent behaviour that can be produced by executing well-defined algorithms (however complex) is in the range of computational intelligence. This includes by definition everything that machines as technological objects (and not just as mathematical abstractions) are already capable of doing, e.g. driving a car, diagnosing medical conditions and proposing an appropriate cure. It excludes everything that is not (yet) fully algorithmically definable, like taking a moral decision. Some will object to the starting point of this discussion: following a rule requires no intelligence. Firstly, the notion of computational intelligence is described in terms of rules; as such, no claim is made below concerning abilities like intuition, creative inspiration or value ascription unless they can be reduced to an algorithmic counterpart. Our only claim is that these do not fall under machine definable behaviour. Secondly, the given definition of computational intelligence is not constrained to rule following, but it includes rule design as well. As explained below, we focus on higher-order algorithmic knowledge in terms of the ability to design and perform algorithms. Designing and executing complex rules that include discerning among huge amount of data in real time, branching over decision trees and performing probabilistic inferences does certainly lead to behaviour that one can describe as intelligent, and hence computational intelligence can at least satisfy the reductionist notion of AGI.

This type of higher-order intelligence is crucially based on the entity's ability to represent knowledge states. Knowledge states are collected in bases (like databases of a machine, or some propositionally encoded version of a human being's experience) equipped with relations and operations to combine and access them in some logical and operationally useful way. The ability of an entity to access its internal states does not necessarily need to be identified with consciousness, although it has been in much philosophical literature. We consider this an avoidable and harmful anthropomorphic

[7] For the historical roots of the analogy between 'programming language' and 'mathematical machine', see [20].

approach to AI, rejected in most part of current technical work. In line with our minimalistic take on intelligence, we shall interpret higher-order intelligence only as the entity's ability to design procedures for the satisfaction of tasks, or the obtaining of state of affairs. Accordingly, we will avoid talking about consciousness, and will just refer to iteration of algorithmic knowledge, or *algorithmic iteration* in short. With this term we refer to the logical principle underlying large areas of current research in theoretical computer science, usually referred to as *automatic algorithm design* (or *automatic programming*), and recently boosted by the creation of genetic and evolutionary algorithms and the application of machine learning techniques. In this sense, it is essential to stress that we are not referring simply to the design of algorithms that include an iteration or recursion principle. Algorithmic iteration represents instead an essential principle of computational intelligence: while rule execution might be identified as trivial, rule design is certainly not. Algorithmic iteration allows to distinguish between execution (of possibly highly complex algorithms) and their design. For the present purposes, the combination of design and execution of algorithms shall suffice to qualify computational intelligence. To represent such reflexive knowledge we will make use of the machinery of modal logic.

The main aim in analysing algorithmic iteration is to reconsider the defining conditions for AGI. In particular, defining a common epistemic state between humans and machines, we wish to investigate whether computational intelligence as displayed by the latter can significantly deviate from the one shown by the former. A logical analysis of this relation allows to critically assess the theoretical possibility of an artificial intelligence so superior to human intelligence to escape its comprehension and control. A further qualification on the validity of our formal results should be stressed: the notion of AGI of relevance to this analysis is confined to the development of programming techniques, i.e. our algorithmic view does not offer a vantage point to consider other approaches such as brain-emulation or mind-emergence from networks, unless the latter can be reduced to a programmable system.

The paper is structured as follows. In section 2 we offer an overview of some significant philosophical and formal literature related to the notion of introspection considered in this paper. In section 3 we consider the philosophical principle relevant to the distinction between first order knowledge interpreted as algorithm execution and higher-order knowledge interpreted as algorithm design. In section 4 we introduce the basic formalism needed for our analysis and consider its informal interpretation. In section 5 and 6 we explore, respectively, the positive and negative versions of human and machine algorithmic iteration. In section 7 we consider some limits of our idealization. In section 8 we consider the problem of the implementation of computational intelligence and link our argument to existing literature on computing mechanisms and the philosophy of computer science. We conclude in section 9 with some observations and further directions of this research.

2 Iteration for Humans and Machines

Self-knowledge, self-awareness or introspection are among the most discussed notions in philosophy.[8] The closely related and philosophically well-known paradoxical counterpart is the notion of self-reference.[9] Since the Liar paradox, this notion has played

[8] For an introductory overview, see [34].

[9] For an introductory overview, see [13].

a huge role in the philosophical tradition and its intersection with formal logic and languages is well known, from the Gödel's sentence to Tarski's theorem. A full review of this large literature exceeds the scope of the present contribution. Nonetheless, our understanding of algorithmic iteration can be qualified in view of some positions on introspection that have been defended both in the philosophical and technical literature.

Several varieties of theories of introspection are present in philosophy of mind and epistemology. Each includes a number of essential and specific conditions to define introspection.[10] The well-known *hard problem of consciousness* from [17] characterizes the philosophical holistic approach to understanding the feeling of awareness of sensory information at any time, as opposed to the *easy problem* of explaining the cognitive experience related to individual experiences, such as object discrimination. In terms of the reductionist approach underlying Definition 1, the philosophical position of interest for our analysis of iteration in AI can be constrained to a version of the *easy problem*:

Definition 3 (Easy Problem of Consciousness (Algorithmic version)) How it is possible to explain higher-order knowledge of algorithmically definable knowledge states (or equivalently, of algorithmically definable procedures for the obtaining of states of affairs)?

Under such a perspective, a major player in this debate has been the *inner-sense theory*, which maintains that knowledge of one's mental life resembles perception, although not being directed towards the current environment or bodily state [4]. This position has been rejected [75], weakened by logical conditions [67], by including perceptual activities [23] and by the use of transparent rules [16]. The latter approach supports the algorithmic version of Definition 3 above and can be briefly recounted as follows:

- there is a fact of the world or property p that is the content of a knowledge state $K(p)$;
- there is an explicit epistemic rule r, embedded in our cognitive abilities and hence to which we have privileged access, that allows iterative knowledge in the presence of a knowledge state;
- $r(K(p))$, hence $K(K(p))$.

This amounts formally to the fragment $KT4$ or $S4$ of modal logic, which we will make use of in the following of this paper: distribution by K axiom is just a property of the implication relation; by the T axiom, objects of knowledge are objects of reality (or truths); and by axiom 4 introspective beliefs follows deductively from already possessed knowledge. This fragment validates also the Necessitation Rule, by which any valid statement in the system of reference is known.[11]

Theories of consciousness from the philosophical debate, asking how is it possible that information processing generates consciousness, have progressively migrated to computational theories on machine consciousness, where the question is to determine which properties of a silicon-based entitiy can generate consciousness.[12] This large and much criticised field has grown in the last twenty years, and recently four different research areas have been identified that associate consciousness to distinct elements [32] :

[10] See [66] for a comprehensive overview.

[11] The relevance of axiom 5 for negative introspection will be considered in Section 6.

[12] See e.g. [2] for an overview.

- External behaviour;
- Cognitive characteristics;
- Architecture.
- Phenomena.

Below we will defend an approach to computable functions as deductively accessible, and in turn to algorithmic knowledge. An entity capable of instantiating the logical principle of iteration for (computationally defined) beliefs is capable of computational intelligence, as per Definition 2. This approach is not entirely new in the more technical literature.

In the formal logic literature, self-knowledge is usually identified as knowledge about beliefs, intentions and desires. Formalization of self-knowledge in first-order logic is known to lead to trivialising inconsistencies [10, 11] as self-referential sentences become expressible. This results either in the design of formal models of first-order logic corresponding to introspective agents with limited deductive power, see e.g. [51], or in the choice of translating the formal apparatus to a modal setting. Modal languages and specific fragments thereof allow to formalise tasks and problems concerning machine intelligence in epistemic, temporal or otherwise defined setting. This, in turn, can be directed towards stratification of knowledge operators, quantification over them (e.g. in first-order modal logic), or dynamic operations with such modalities, e.g. to express how knowledge states change in view of new information becoming available (typical of the dynamic epistemic logic approach with public announcement operators), useful for planning agents [12]. Other areas of applied computational research in which introspection is playing an important role are: collection of incomplete information in databases [33], multi-strategy learning [18], index refining [29], intelligence analysis [36], meta-cognition for augmented data-analysis [68], interaction of automated agents with tacit knowledge [22] and several more.

The bridge from formal logic to applications in computer science concerning iteration of algorithmic constructions is formulated in terms of automatic algorithm design and automatic programming. This area can be traced back to compiler development for high-level languages in the ′50s, although proper research started only in the ′80s. The initial research was adamant about the need to extract essential principles of human design involved in algorithm specification and implementation: problem understanding, solution planning and refining, algorithm execution, limits and opportunities identification, solution verification and evaluation, see e.g. [47], [48], [1]. In this early research, algorithm design was defined as the process of producing a computationally feasible (i.e. computable in polynomial time), relatively complete and consistent sketch of the task to be accomplished from a given specification. Program synthesis is the associated process of choosing data structures and functions to transform the algorithm in code. Notice that the ability to acquire in a partially automatic way the high-level specification to be compiled are essential even at this early stage, see e.g. [6]. In the ′80s, automation work left to the machine was variable, depending on the level of natural language involved, the amount of informality allowed and the distinction between being able to tell the machine *what* to do, rather than *how* to do it. Formal methods were involved in this process by helping synthesizing programs from formal specifications and from examples, [9]. In parallel, the aspects related to building knowledge domains were gaining relevance [8]. The rise of new computational models over the ′90s and 2000s, e.g. networks, distributed algorithms, web search and link analysis, and later the development of randomized and evolutionary algorithms, have brought algorithm design

problems to a whole new level.[13] Today, automatic programming is for a large part a process of optimization with a large number of design choices, parameters, non-linear interactions for both exact and approximate algorithms. It requires the application of search techniques, heavy computational power to explore design spaces and it involves stochastic aspects in exploring instances of the problem observed, while learning processes are essential to solve unseen instances of the same problem. The requirements written in a formal specification are usually transformed sequentially into low-level implementations, a process automatized by genetic programs that evolve the candidate for transformation [3].

The algorithmic approach to the easy problem of consciousness identifies in (possibly automatic) algorithm design the mechanical ability of accessing rules for mechanical execution. We shall offer a reading that is neutral with respect to the entities of relevance through a modal logic translation. The use of modal logic, beyond modeling several forms of human rationality, is a well-established technique to reason about programs as well since the '70s [61,25], codified by Hoare Logic [45] and more recently by Dynamic Logic [41]. In the setting of the latter system, the modal formula $[\alpha]p$ says that after execution of program α, p holds. Iteration is used to analyse execution of sequential programs, including those which loop on their own execution: $[\alpha;\alpha]p \rightarrow [\alpha][\alpha]p$ says that if after execution of program α which loops on itself, p holds, then after execution of α one reaches a state where after execution of α, p holds. The general interpretation of the iteration axiom schema of modal logic for both humans and machines suggested in the present work is possible under the so called *proofs as programs isomorphism*, also known as Curry-Howard-de Bruijn isomorphism [19,46,15,69]: extensionally, given a proposition A provable by proof a, there is a program p that satisfies the specification S corresponding to A

$$a : A \leftrightarrow p : S$$

This principle states that for every algorithmically executable process to prove a proposition – and hence realizing the corresponding state of affairs – there is a correct program that satisfies the same state of affairs. This principle expresses the conceptual identity between effective computability and mechanical realizability. Once the iteration principle is applied, it follows the conceptual identity between algorithm design for machines and epistemic rule access for humans.

Our account of algorithmic iteration below is therefore based on an interpretation of introspection through transparent rules introduced earlier, although we stress some specific properties:

1. algorithm execution to the effect that there is an entity knowing that p is neutral with respect to the nature (human or mechanical) of the subject executing the algorithm, according to the above introduced proofs-as-programs isomorphism;
2. the iterative step $K(K(p))$ is algorithm design, in line with automatic programming and epistemic rule access.

This interpretation, detailed in the following section, relies on a *de dicto/de re* distinction on modalities, with several practical implications relevant to the design of computational agents in AI.

[13] See e.g. [49] for a modern overview of the discipline.

3 Design and Execution

The interpretation of syntactic terms as algorithmic operators and their outputs induces immediately a question concerning their scope. This recalls the old-standing issue related to propositional attitudes and the famous *de re/de dicto* distinction. This distinction has been considered in its syntactical, semantical and metaphysical interpretation, see [53]. The distinction is famously embedded in the Barcan formula [7] in quantified modal logic:

$$\forall x \Box \phi x \rightarrow \Box \forall x \phi x \tag{1}$$

which is notoriously equivalent to

$$\Diamond \exists x \phi x \rightarrow \exists x \Diamond \phi x \tag{2}$$

An informal explanation of these formulas can be given by saying that by 2 nothing comes into existence, by 1 nothing goes out of existence. The philosophical implications of these formulas are well known, especially with respect to the thesis known as *actualism* and their relevance for the metaphysical justification of contingent objects.[14] The obvious problematic aspect of 2 is that not everything that is plausible to assume can exists has to actually exists.

In the present context we are interested in an interpretation of the *de re/de dicto* distinction applying to algorithms. An attempt in this direction was already presented in [55], [57], where a formalism is developed to account for the sense or intension of an expression as an algorithm, and algorithm execution determining denotation. In a similar vein, recent philosophical accounts of information processing for digital computation use the notion of instructional information to account for the procedural sense of the computation, as opposed to its denotation as output, see e.g. [30]. Our main concern is the qualification of the formula in the scope of the modal operator intended as an algorithm. In its first interpretation, where a non-modal formula is in its scope, we propose a *de re* reading which amounts to *algorithm execution*:

Definition 4 (Algorithmically *de re*) An algorithmic evaluation is *de re* with respect to an output specification S if and only if it directly points to the *execution* of the algorithm or program p producing an output satisfying S.

This means that given a modal formula $\Box A$, where A is non-modal, its algorithmically *de re* interpretation makes it correspond to a term formula $p : S$, with program p and specification S such that the execution of p satisfies S, and S is the specification that corresponds to A.

In its second interpretation, where a modal formula is in the scope of an operator qualified as an algorithm, we propose a *de dicto* reading which amounts to *algorithm design*:

Definition 5 (Algorithmically *de dicto*) An algorithmic evaluation is *de dicto* with respect to an output specification S if and only if it indirectly points to S, i.e. by pointing at the algorithmic *design* of the algorithm or program p producing an output satisfying S.

[14] For an overview of this debate, see [54].

This means that given a modal formula $\Box(\Box A)$, where A is non-modal, the algorithmically *de dicto* interpretation of the outermost $\Box$ makes it correspond to a term formula $p' : (p : S)$, with program p' such that its execution satisfies the design of a program p whose execution satisfies S.

In view of the above *de re/de dicto* distinction concerning algorithms, in the following we denote a modal expression in the range of a modal operator as *algorithm design*, and a non-modal formula in the scope of a modal operator as *algorithm execution*.

4 Formal Preliminaries

We use an alphabet of atomic propositions to denote the encoding of elements of a knowledge base. Such elements can be thought of as states of affairs, actions, objects of desires or of beliefs. Atoms can be used to construct more complex formulas. In line with our algorithmic view, we restrict here ourselves to a fragment of first order logic including modalities which is a reduction class in the sense that a computable function for the satisfiability of formulas exists. Informally, this means we restrict to formulas expressing some specification for an algorithmically executable process (program), i.e. we assume that computable functions can be given for all valid formulas or tautologies. The language

$$\mathcal{L} := \{\phi \mid \neg A \mid A \to B \mid \Box_i A\}$$

with ϕ in the set of atoms and A, B metavariables ranging over formulas, is some decidable fragment including at least propositional modal logic. The index on modal formulas is explained below. We can further extend the language with additional connectives, but this is not required for our purposes.

We read A simply as 'A is the case'. Based on the distinction of modal operators introduced in the previous section, we read the necessitation formula $\Box A$ as 'an algorithm is executed to the effect that A is the case'. This reading reflects the *de re* interpretation of the modal operator: it is a necessary feature of the computational ontology of our world that executing a given algorithm produces a certain output A. This relies further on the assumption that the logical structure of the algorithm, its implementation and execution are correct. Hence a boxed formula refers to a specification satisfied by an algorithm.

We further characterise our modal formulas introducing subscripts with the following meanings:

$\Box_H A$: 'an algorithm is executed by a human to the effect that A is the case';

$\Box_M A$: 'an algorithm is executed by a machine to the effect that A is the case'.

The meaning of the standard T axiom from modal logic is then given in the following two versions:

(T_H) $\Box_H A \to A$: 'If an algorithm is executed by a human to the effect that A, then A is the case';

(T_M) $\Box_M A \to A$: 'If an algorithm is executed by a machine to the effect that A, then A is the case'.

The former expresses algorithm execution by humans and it can be understood as an informal counterpart to Church's notion of effective computability. The latter expresses algorithm execution by machines and it can be understood as an informal counterpart to Turing's notion of mechanical realizability. Admitting this axiom makes our algorithmic interpretation stronger than the typical provability interpretation for modal logic, e.g. in the system **GL** where neither axiom T nor axiom D are admissible as the reference system might be unsound or inconsistent. In the present algorithmic interpretation, the intended meaning of the modal operator is the existence of a feasible, executable procedure satisfying a given output. We exclude non well-defined, non-terminating algorithms from our analysis, restricting ourselves to programs whose formal correctness with respect to the specification can be proven in principle (and therefore to decidable formulas).

We moreover admit counterparts to the K axiom for distribution of $\Box$ over implication. Our analysis in the following shall focus on the meaning of so-called iteration axioms (Axiom 4) for modal logic. As anticipated in the previous section, when a boxed formula is in the scope of another modal operator and the iterative step takes place, we assign a different reading to it so as to reflect its different range (which is no longer a specification, but an algorithmic action). This *de dicto* reading is then interpreted as algorithm design:

> $\Box_X \Box_H A$: 'an algorithm is *designed* by X to the effect that an algorithm is executed by a human to the effect that A';
>
> $\Box_X \Box_M A$: 'an algorithm is *designed* by X to the effect that an algorithm is executed by a machine to the effect that A'.

We formulate now the proofs-as-program principle observed in section 2:

Definition 6 (Human-Machine Algorithmic Correspondence) $\Box_H A \leftrightarrow \Box_M A$

The intended reading of the proofs-as-programs equivalence is that if an algorithm to the effect that A is executed by a human, then one to the same effect is executed by a machine, and vice versa. Albeit equivalent, the two algorithms are not the same:

Definition 7 $\Box_M A$ is a more efficient and optimal version of $\Box_H A$.

Notice that our analysis does not prescribe that any machine designed algorithm is the better counterpart of an *existent* human designed algorithm, or of one that would otherwise be designed sooner or later by a human. Definition 7 says instead that a machine executed algorithm (however unintended or unexpected by a human designer, e.g. because of being the result of learning huge amount of data and drawing correlations that are difficult to discover at first) is never a procedure that remains inexplicable or out of the control of a human designer.

As pointed out above, the fragment of modal logic considered in our analysis is restricted to decidable formulas. This choice is justified by our interest in specifications with programs, and formally by the fact that a logic including $S4$ is bound to validate the necessitation rule: 'If A is the case, then $\Box A$'. In the semantics this rule states that for every tautology there is an algorithm to verify it, which amounts to the problem of logical omniscience in an algorithmic version. This old philosophical problem [44, 71] has been already approached under an algorithmic version at least since [40], where an interpreted system is offered where agents have algorithms as part of their local

knowledge states, and hence algorithmic knowledge is explicit but distinct from validity. As such, no agent algorithmically knows all tautologies, nor all the consequences of her own algorithmic knowledge. Instead, it is possible to prove in this system that any agent has (implicit) knowledge of her own algorithmic knowledge. We follow here a different strategy to accommodate (not to solve) the issue of algorithmic omniscience: we constraint the system to validities that are algorithmically definable, and as such are at least in theory consistent with the algorithmic interpretation of necessitation.

5 Positive Algorithmic Iteration

In this section we analyse and interpret the various versions of Axiom 4 possible by interleaving distinct indices on our modalities. We list their meaning and formulate some obvious consequences. In the following, the neutral assignment X will be substituted by H and M respectively. With respect to these qualifications of algorithms, it is essential to keep in mind the level of abstraction we are considering: when referring to $\Box_H A$ as "an algorithm is executed by a human to the effect that A", we refer to an idealised model of effective computability in terms of a human computor in the process of going through the required steps for obtaining the state of affairs encoded by A as a proposition; similarly, by $\Box_M A$ intended as "an algorithm is executed by a machine to the effect that A", we refer to a mathematically correct model of a physical machine in the process of going through a step-by-step set of instructions, i.e. a model of mechanical realizability.

Let us now list all the combinations of human- and machine-designed algorithms for human- and machine-executed algorithms without negation.

5.1 On human algorithms

$$4_{HHH} \qquad \Box_H A \rightarrow \Box_H \Box_H A$$

If an algorithm is executed by a human to the effect that A, then an algorithm is designed by a human to the effect that such an algorithm exists.

$$4_{HMH} \qquad \Box_H A \rightarrow \Box_M \Box_H A$$

If an algorithm is executed by a human to the effect that A, then an algorithm is designed by a machine to the effect that such an algorithm exists.

$$4_{HHM} \qquad \Box_H A \rightarrow \Box_H \Box_M A$$

If an algorithm is executed by a human to the effect that A, then an algorithm is designed by a human to the effect that an equivalent algorithm executed by a machine exists.

$$4_{HMM} \qquad \Box_H A \rightarrow \Box_M \Box_M A$$

If an algorithm is executed by a human to the effect that A, then an algorithm is designed by a machine to the effect that an equivalent algorithm executed by a machine exists.

Lemma 1 (Human Algorithmic Completeness) *Everything that is algorithmically feasible for a human to execute, can logically be designed and executed by both humans and machines.*

Proof By $4_{HHH}, 4_{HMH}, 4_{HHM}, 4_{HMM}$.

Lemma 2 *Everything that is algorithmically feasible for a human to execute, can logically be designed in a faster and optimal form by a machine.*

Proof By 4_{HMH} and Definition 7.

Lemma 3 *Turing Machines are a logical possibility.*

Proof By 4_{HHM}.

The results from this section justify the thesis that mechanically executed computational intelligence can level with – and in fact improve on – any humanly executable act of intelligent computation.

Lemma 4 *Turing Machines can optimise algorithms executed by humans.*

Proof By Definition 7, Lemma 3 and 4_{HMM}.

5.2 On machine algorithms

$$4_{MMM} \qquad \Box_M A \rightarrow \Box_M \Box_M A$$

If an algorithm is executed by a machine to the effect that A, then an algorithm is designed by a machine to the effect that such an algorithm exists.

$$4_{MHM} \qquad \Box_M A \rightarrow \Box_H \Box_M A$$

If an algorithm is executed by a machine to the effect that A, then an algorithm is designed by a human to the effect that such an algorithm exists.

$$4_{MMH} \qquad \Box_M A \rightarrow \Box_M \Box_H A$$

If an algorithm is executed by a machine to the effect that A, then an algorithm is designed by a machine to the effect that an equivalent algorithm executed by a human exists.

$$4_{MHH} \qquad \Box_M A \rightarrow \Box_H \Box_H A$$

If an algorithm is executed by a machine to the effect that A, then an algorithm is designed by a human to the effect that an equivalent algorithm executed by a human exists.

Lemma 5 (Machine Algorithmic Completeness) *Everything that is algorithmically feasible for a machine to execute, can logically be designed and executed by both humans and machines.*

Proof By $4_{MMM}, 4_{MHM}, 4_{MMH}, 4_{MHH}$.

Lemma 6 *Every machine can design non-optimal programs.*

Proof By Lemma 5.

Lemma 7 (Universal Turing Machine) *The Universal Turing Machine is a logical possibility.*

Proof By Lemma 5.

The results from this section justify the thesis that humans can level with any machine executable act of intelligent computation (modulo enough time and space). Or in other words, any algorithmic limit to human computational intelligence is one for machine computational intelligence as well. The following lemma is the main result:

Lemma 8 *Humans can control algorithms designed and executed by machines.*

Proof By Definition 6 and 4_{MHM}.

5.3 At the border

Collating the results on positive algorithmic iteration from the previous subsections, we offer a principle of completeness and one of transparency.

Theorem 1 (Human completeness over Machine Algorithms) *Given enough time and modulo non-optimality, a human can design and modify any algorithm designed by any machine.*

Proof By Definition 7 and Lemma 8.

Theorem 2 (Non-opacity of computational intelligence) *For every propositionally encoded state that can be algorithmically executed by either a human or a machine, there are human (non-optimal) and machine (optimal) designs transparent to each other.*

Proof By Lemmas 2 and 8.

6 Negative Algorithmic Iteration

The extension of the logic $S4$ to iteration with negation is obtained through axiom 5,

$$5 \qquad \neg\Box\neg A \rightarrow \Box\neg\Box\neg A$$

The standard epistemic interpretation of $\neg\Box\neg A$ is given by equivalence to the possibility operator $\Diamond$. In the present context of an algorithmic interpretation of the iteration axioms, an appropriate reading of negation is required.

We start by considering $\Box A$ to be corresponding to a characteristic function for A, i.e. a function with Boolean outputs returning 1 or `True` for an algorithm that satisfies A, and 0 or `False` otherwise. This makes an algorithm for A computable in polynomial time P, the class of all decision problems solvable by a deterministic Turing machine using a polynomial amount of computation time. Accordingly, $\Box\neg A$ is also a problem computable in P. Then $\neg\Box\neg A$ becomes a problem that is not computable in polynomial time ($\overline{P}$). This class contains everything that is not computable by a deterministic algorithm running in polynomial time. The class NP of decision problems where a positive answer can be verified in polynomial time by a non-deterministic Turing-Machine contains P but is generally believed not to coincide with it. Above NP, $PSPACE$ is the set of all decision problems that can be solved by a Turing Machine using polynomial space and $EXPTIME$ is the set of all decision problems that have exponential runtime.

$\Box_{\overline{P}}A$ can now be used to express any algorithm whose execution requires time and computational resources that are beyond the standard computational abilities of the appropriately chosen (mathematically idealised) model of human intelligence. As they are above our idealised model, they certainly are above the limits of any real, resource-bounded agent. We can identify in this way hard algorithmic problems in terms of computational intelligence. Negative iteration amounts then to the following two formulations:

$$5_H \qquad \Box_{\overline{P}}A \rightarrow \Box_H\Box_{\overline{P}}A$$

If a hard algorithm is executed to the affect that A, then there is a human-design for it.

$$5_M \qquad \Box_{\overline{P}}A \rightarrow \Box_M\Box_{\overline{P}}A$$

If a hard algorithm is executed to the affect that A, then there is a machine-design for it.

Notice that the antecedent only claims that such an algorithm is executed, assuming it exists: the existence itself is an undecidable problem. Examples of hard algorithms include design of experiments in statistics, protein design in biology, graph colouring and path construction in geometrical complexity, string-to-string correction in programming, optimization in machine learning. Despite these algorithms being 'hard' in a practical and procedural sense, their design and eventually their understanding is still within the scope of computational intelligence (modulo time and space).

7 Fulfilment and Faithfulness

In [50] a computational theory of belief introspection is developed aimed at analysing the nature of computational introspection on machines with a non-doxastic knowledge base. Our analysis has a different take, in that it aims at revealing the expressive possibilities of a modal language when the modality is taken to interpret an algorithmic procedure and the iteration allows for both human and mechanical agents' interleaving. Despite this dissimilarity, a crucial aspect of the work in [50] can be adapted to our purposes, namely the results on introspection for perfect and bounded rational

agents. What follows can be considered a purely formal adaptation. We start by a generalization of the results from the previous section.

$$\Box_X \Box_Y A \leftrightarrow \Box_Z A \quad X, Y, Z \in \{H, M\}$$

This principle can be proven to hold by assuming consistency of $\Box_X A$ and completeness over such algorithms, i.e. that for any action or state A, either the design of an algorithm to the effect that $\Box_X A$ exists, or one can prove that no such design is possible. A formal limitation by undecidability is still in place, as $\Box_X A$ cannot be in general shown to hold for every A. The right to left direction of this equivalence is just a combination of the various iterations of axiom 4 and it is called *positive fulfilment* in [50]. Its negative counterpart extracted from axiom 5 is called *negative fulfilment*.

Proposition 1 (Fullfilment)

$$\Box_Z A \rightarrow \Box_X \Box_Y A$$

The left to right direction of the equivalence is called *faithfulness* in [50]. It expresses the ability of an iterative computational system to design an algorithm to the effect that A if its non-iterative counterpart can execute an algorithm to the effect that A (and not do so if such a performance is not available). Under our interpretation for negative iteration from Section 6, the principle has the following form:

Proposition 2 (Faithfulness)

$$\Box_X \Box_Y A \rightarrow \Box_{\overline{P}} A$$

It expresses the fact that if an iterative computational system (human, machine or interleaving the two) can be shown to design an algorithm to the effect that A, then such an algorithm must exists in some complexity class at most over P.

8 Computational Intelligence is implementation neutral

Our argument has considered the formal identity of algorithm design across mechanical computation (in the sense of Turing) and effective computability (in the sense of Church). For the former we have referred to rule-based introspection, i.e. the ability to access internal procedural rules to formulate step-by-step actions; for the latter, we considered automatic programming, now increasingly based on new machine-learning techniques reliant on efficient categorization and probabilistic assessment. The result expressed by Theorem 2 establishes that automatically designed algorithms are transparent to human understanding, modulo optimality.

It is a largely accepted view, in biology as well as in psychology and AI, that intelligence is an essentially embodied property. Here we do not discuss the extensive problem of determining whether different embodiments (e.g. in mechanical or biological components) change in any significant way the nature of intelligence. Instead, we are interested in showing that fixing the notion of computational intelligence in the sense of Definition 2, any embodiment preserves it. In the terminology introduced above in Section 1, we need to show that algorithmic knowledge for mathematical objects of infinite size is preserved not only by mathematical objects of finite size, but also by

technological objects. In other words, we need to show that the identity of algorithmic iteration between humans and machines is neutral with respect to implementation and hence the result of Theorem 2 holds irrespective of the entities manifesting computational intelligence. To show that this is the case, it is essential to illustrate how such identity is preserved at each level of abstraction in the ontology of computing mechanisms.[15] The multi-layered analysis of computational systems, reliant on the practice of computer scientists and philosophically advocated since at least [26], is essential to the mechanistic explanation of physical computing [60], as well as to the identification of their malfunctioning [31,28]. For the present purposes, it is sufficient to apply the analysis of abstract information for computational systems offered in [63]. In that context, it is shown how each level in the ontology of computing mechanisms has an associated epistemological explanation and their relation is offered in informational terms. In the present context we show that any structure implementing computational intelligence qualifies as a computing mechanism in the same terms.

We start our description of computing mechanisms at the lowest abstraction level, where the ontology is expressed as *physical data*. The associated epistemological interpretation is the realization of *actions* by hardware. Here information consists of just *structured data*. To exemplify, structured data for mechanical computing is difference-making electrical charges; in biological structures, and in particular in humans implementing rule-based operations, it is a well-defined topology of firing neurons in a network.[16] The next higher level is the *encoding of physical states*, whose associated epistemological interpretation is the performing of *operations* as control of actions. Information at this level is *well-formed, performative data*: for (digital) computational systems this corresponds to machine low-level languages, for humans this is the instruction encoded in DNA and related functions. The ontology at the next higher level is *encoded operations* and the corresponding epistemological structure is the definition of *instructions* as control of operations. This can be expressed as instructional information in terms of *syntactically well-formed, meaningful data*: in machines it is expressed by programming languages, in humans is reflected by natural language instructions. One level higher up, the ontology of the computing mechanism is the logic of *algorithms*, with its epistemological explanation given by the expressions of *tasks*, to be instructed, performed and executed. Here information is *abstract and correctness-determining data*, i.e. establishing which instructions (and hence operations and actions) are correct in view of the current aims. This level is already neutral for machines and humans, as expressed by the formal identity between mechanical computation and effective computability. Finally, the highest level of the ontology of computing mechanisms is that of the *intention state*, with the associated epistemological explanation given by *problem solution*. This duality determines algorithm design, i.e. the iteration step specifically considered in this paper. At this level, content is characterised as *abstract, semantically loaded, truth-determinant information* for the algorithm to be designed. At this level, one asks what the intended action of algorithmic intelligence is, and the latter in turns defines the models where some actions can be considered true realisations of that intention, others will not. This level is again neutral, as shown by Theorem 2.

This brief recount suggests that an ontology implementing computational intelligence as by Definition 2 proceeds in the terms described above: problem formulation, task assessment, instruction encoding, operation encoding, action execution. We argue

[15] We borrow the term *computing mechanism* in the technical sense introduced by [59].

[16] Notice that determining which topology is associated with which action is a hard problem.

that any entity whose function of interest is that described by computational intelligence according to the description above is a computing mechanism, irrespective of its implementation. This description is only functional, in the sense that it does not exclude that the same ontology might be able to qualify differently with respect to other functions: in particular, a human intelligence will be epistemologically different when it does not perform acts of computational intelligence in this specific sense. This description is objective in the sense of the mechanistic account [60], as it qualifies as computational only those activities that have a well-defined task and an algorithmic solution to it. Moreover, the explanation provided reflects the essence of the mechanistic account because it relies on identifying components at each abstraction level, with the associated function and in relation to other components (i.e. the organization). Here the mechanistic account of reference is by definition restricted to recursive functions, encoded in a relevant way for the appropriate ontology. The complementary question whether an artificial entity can produce intentions that cannot be produced by its biological counterpart would first require an epistemological description of such ontology that does not qualify it as a computing mechanism: we have argued that such is not the case for the technology mastered so far.

9 Conclusions

We have presented a modal argument on the possibility of iterative computational systems. We have formally argued that interleaving of human and machine algorithms at design and execution stages is possible: such an argument justifies transparency of computational intelligence to both humans and machines. One main consequence of such an argument is that algorithmic design and execution by a machine can in principle always be translated to an equivalent algorithm designed and execution by humans, modulo efficiency in retrieving and computing resources. These results are embedded in the principles of Fulfilment and Faithfulness. We have explicitly constrained our analysis to specification with associated provably correct algorithms.

The current debate on Artificial Intelligence has developed in particular towards the theoretical justification that an artificial agent might develop computational abilities which, through some form of self-reflection, can progress to stages that largely surpass any human intelligence. This has ignited a heated debate over the possibility that such a Super-Intelligence might develop objectives that are non-aligned to human principles and in turn become dangerous to our species. Our formal argument can be interpreted as offering a strong theoretical basis for reconsidering the arguments on the (ethical, economical, epistemological) risks of the technological singularity when this is expected to result from current technologies based on programming. We believe that fulfilment and faithfulness express the current reality of artificial intelligence: what machines can do, is exactly what we program them for and our current understanding of machine behaviour is under control in the sense that we know the range of expected outputs from correctly behaving programs, although we might not ourselves be able to produce those outputs. It should be stressed moreover that failure and errors in open systems are sources of unbounded risks. This part of our thesis is relatively trivial and unsurprising.

Beyond this, faithfulness expresses a stronger theoretical position. It claims that any self-programmable machine will still operate in a range of input-output relations theoretically accessible to humans. This certainly does not mean that humans will always be able to anticipate and hence control the actions of an artificial agent endowed

with some level of intelligence: the speed and sheer amount of data that can be organised, controlled, analysed for correlations and analogies and its rule-inference power will grow beyond our finite abilities. But whatever such machine might be able to learn and plan to do is still within the limits of algorithmic design and understanding: these are ultimately under our control. This thesis is strengthened by showing that any implementation of Computational Intelligence amounts to a computing mechanism.

We believe that safe Artificial Intelligence is a task for designers, who need to put in place safe-lock procedures to allow humans to rely on its use. The design of safe AI is one of the available views on required responses to catastrophic AGI risks, see [70] for an overview, [42] for a recent model of safe AI design. Computational intelligence supports the thesis that AI research should enforce internal constraints for both technically well-functioning algorithms and philosophically well-motivated aims, [73]. The possibility to develop highly beneficial AI is at our disposal and this is the time to draw the appropriate limits and ensure that we start from the right principles.

Acknowledgments

Thanks to Raf Pasmans, who unexpectedly prompted my first thoughts on this topic. Thanks to the participants at the 'Debate on Moral and Philosophical Aspects of Artificial Intelligence' at Ghent University and at the 'Conference on Artificial Intelligence and Contemporary Society: The Role of Information' at University of A Coruña, where early versions of this paper were presented. I am especially grateful to Wenceslao Gonzalez, Luciano Floridi, Patrick Allo and Nikos Gkorogiannis, whose observations and comments helped address some conceptual and formal issues. Two anonymous referees' critiques have helped improving the final version of this paper.

References

1. Adelson, B., Soloway, E. A model of software design. *International Journal of Intelligent Systems*, 1(3):195–213, 1986 (republished 2007).
2. Aleksander, I.L. Machine Consciousness. In M. Velmans & S. Schneider (eds.), *The Blackwell Companion to Consciousness*, pp.87-98, 2007 .
3. Arcuri, A., Yao, X. Co-evolutionary automatic programming for software development. *Information Sciences*, 259:412-432, 2014.
4. Armstrong, D.M. *The Nature of Mind and other Essays.* Cornell University press, Ithaca, N.Y, 1981.
5. Baeten, J.C.M., Luttik, B. and van Tilburg, P.J.A. Reactive Turing machines. *Information and Computation*, 231:143–166, 2013.
6. Balzer, R, A 15 Year Perspective on Automatic Programming. *IEEE Transactions on Software Engineering*, 11(11):1257–1268, 1985.
7. Barcan (Marcus), R. C. A functional calculus of first order based on strict implication. *Journal of Symbolic Logic*, 11:1-16, 1946.
8. Barstow, D. A Perspective on Automatic Programming. *AI Magazine*, 5(1):5–27, 1984.
9. Biermann, A.W. Automatic Programming: A Tutorial on Formal Methodologies. *Journal of Symbolic Computation*, 1:119-142, 1985.
10. Bolander, T. Maximal Introspection of Agents. *Electronic Notes in Theoretical Computer Science*, 70(5):183-198, 2002.
11. Bolander, T. *Logical Theories for Agent Introspection.* PhD Thesis, Informatics and Mathematical Modelling, Technical University of Denmark, 2003.
12. Bolander, T., Andersen, M.B.: Epistemic planning for single and multi-agent systems. *Journal of Applied Non-Classical Logics*, 21(1): 9-34, 2011.

13. Bolander, T. Self-Reference. *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta (ed.), Spring 2015 Edition. `http://plato.stanford.edu/archives/spr2015/entries/self-reference/`.
14. Bostrom, N. *Superintelligence: Paths, Dangers, Strategies*. New York: Oxford University press, 2014.
15. De Bruijn, N. *Automath, a language for mathematics*. Department of Mathematics, Eindhoven University of Technology, TH-report 68-WSK-05, 1968. Reprinted in revised form, with two pages commentary, in: *Automation and Reasoning, vol 2, Classical papers on computational logic 1967– 1970*, pp. 159–200, Springer Verlag, 1983.
16. Byrne, A. Introspection. *Philosophical Topics*, 33(1): 79–104, 2005.
17. Chalmers, D. *The Conscious Mind: In Search of a Fundamental Theory*. Oxford University Press, 1996.
18. Cox, M.T., Ram, A. Introspective Multistrategy Learning: On the Construction of Learning Strategies. *Artificial Intelligence*, 112(1-2):1-55, 1999.
19. Curry, Haskell B., Feys, R. *Combinatory Logic Vol. I*. Amsterdam: North-Holland, 1958.
20. Daylight, E.G. Towards a Historical Notion of 'Turing – the Father of Computer Science'. *History and Philosophy of Logic*, 36(3):205-228, 2015. `http://www.tandfonline.com/doi/full/10.1080/01445340.2015.1082050`.
21. De Grave, K. (ed.) *Formalism & Intuition in Software Development. A conversation with Michael A. Jackson conducted by Edgar G. Daylight and Bas van Vlijmen*. Conversations, Issue 5, Lonely Scholar, 2015.
22. van Ditmarsch, H., French, T. On the Interactions of Awareness and Certainty. *Australasian Conference on Artificial Intelligence*, 727-738, 2011.
23. Evans, G. *The Varieties of Reference*. Oxford University Press, Oxford, 1982.
24. Fallenstein, B., Soares, N. Vingean Reflection: Reliable Reasoning for Self-Improving Agents. Technical Report 2015-2, Machine Intelligence Research Institute, 2015.
25. Fischer, M.J., Ladner, R.E. Propositional modal logic of programs. In STOC '77 *Proceedings of the ninth annual ACM symposium on Theory of computing*, 286-294 , 1977.
26. Floridi, L. The Method of Levels of Abstractions. *Minds & Machines*, 8(3):303–329, 2008.
27. Floridi, L. Singularitarians, AItheists, and Why the Problem with Artificial Intelligence is H.A.L. (Humanity At Large), not HAL. *APA Newsletter*, 14(2):7–11, 2015.
28. Floridi, L., Fresco, N., Primiero, G. On malfunctioning software. *Synthese*, 192(4):1199-1220, 2015.
29. Fox, S., Leake, D.B. Introspective reasoning for index refinement in case-based reasoning. *Journal of Experimental and Theoretical Artificial Intelligence*, 13(1): 63-88, 2001.
30. Fresco, N. Information Processing as an Account of Concrete Digital Computation. *Philosophy & Technology*, 26(1):31–60, 2013.
31. Fresco, N., Primiero, G. Miscomputation. *Philosophy & Technology*, 26(3):253–272, 2013.
32. Gamez, D. Progress in machine consciousness. *Consciousness and Cognition*, 17:887-910, 2008.
33. Gelfond, M. Strong Introspection. *AAAI 1991*, 386-391, 1991.
34. Gertler, B. Self-Knowledge. *The Stanford Encyclopedia of Philosophy* (Summer 2015 Edition), Edward N. Zalta (ed.), `http://plato.stanford.edu/archives/sum2015/entries/self-knowledge/`.
35. Gilbert, S. and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Newsletter ACM SIGACT News*, 33(2):51-59, 2002.
36. Goel, A.K., Morse, E.L., Raja, A., Scholtz, J., Stasko, J.T. Computational Explanations for Report Generation in Intelligence Analysis. *ExaCt 2009*:37–47, 2009.
37. Goertzel, B., Pennachin, C. (eds.) *Artificial General Intelligence*, Springer, 2007.
38. Good, I.J. Speculations concerning the First Ultraintelligent Machine. In F.L. Alt and M. Rubinoff (eds.), *Advances in Computers*, 6:31-88, 1965.
39. Gurevich, Y. What is an Algorithm? *SOFSEM 2012: Theory and Practice of Computer Science*, Lecture Notes in Computer Science 7147:31-42, 2012.
40. Halpern, J.Y., Vardi, M. Algorithmic Knowledge. In R. Fagin (ed.), *Proceedings of the 5th Conference on Theoretical Aspects of Reasoning about Knowledge*, pp.255–266, Morgan Kaufmann, 1994.
41. Harel, D., Kozen, D., Tiuryn, J. *Dynamic Logic*. MIT Press, 2000.
42. Hibbard, B. Decision support for safe AI design. In J. Bach, B. Goertzel and M. Iklé (eds.), *Artificial General Intelligence. Lecture Notes in Artificial Intelligence*, 7716:117–25, New York:Springer, 2012.
43. Hill, R. What an Algorithm Is. *Philosophy & Technology*, 29(1):35–59, 2016.

44. Hintikka, J. Impossible possible worlds vindicated. *Journal of Philosophical Logic*, 4:475-484, 1975.
45. Hoare, C.A.R. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12:576-580, 1969.
46. Howard, William A. The formulae-as-types notion of construction. In Seldin, Jonathan P.; Hindley, J. Roger, To H.B. Curry: *Essays on Combinatory Logic, Lambda Calculus and Formalism*. Boston, MA: Academic Press, pp. 479–490, 1980. (original paper manuscript from 1969).
47. Jeffries, R., Turner, A.A., Polson P.G. The processes involved in designing software. In J.R. Anderson (ed.), *Cognitive Skills and their Acquisition*, ch. 8. Hillsdale, NJ: Lawrence Erlbaum, 1981.
48. Kant, E. Understanding and Automating Algorithm Design. *IEEE Transactions on Software Engineering*, vol SE-11, 1985.
49. Kleinberg, J. and Tardos, E. *Algorithm Design*. Pearson Addison-Wesley, 2005.
50. Konolige, K. A Computational Theory of Belief Introspection. In *IJCAI*, 85:503-508, 1985.
51. Lakemeyer, G. Limited reasoning in first-order knowledge bases with full introspection. *Artificial Intelligence*, 84:209-255, 1996.
52. van Leeuwen, J.J and Wiedermann, J. Beyond the Turing limit: Evolving interactive systems. In: Leszek Pacholski, Peter Ruzicka (Eds.), SOFSEM, in: *Lecture Notes in Computer Science*, 2234:90-109, 2001.
53. McKay, T. and Nelson, M. Propositional Attitude Reports. *The Stanford Encyclopedia of Philosophy* (Spring 2014 Edition), Edward N. Zalta (ed.), `http://plato.stanford.edu/archives/spr2014/entries/prop-attitude-reports/`.
54. Menzel, C. Actualism. *The Stanford Encyclopedia of Philosophy* (Summer 2016 Edition), Edward N. Zalta (ed.), `http://plato.stanford.edu/archives/sum2016/entries/actualism/`.
55. Moschovakis, Y. N. Sense and denotation as algorithm and value. In J. Oikkonen and J. Vaananen (Eds.), *Lecture Notes in Logic*, Volume 2, pp. 210–249. Springer, 1994.
56. Moschovakis, Y.N. What is an algorithm? In Engquist, B., Schmid, W. (eds.) *Mathematics Unlimited - 2001 and Beyond*, pp. 919–936. Springer, 2001.
57. Moschovakis, Y. A logical calculus of meaning and synonymy. *Linguistics and Philosophy*, 29:27–89, 2006.
58. Nilsson, N. *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press, 2009.
59. Piccinini, G. Computing Mechanisms. *Philosophy of Science*, 74:501-526, 2007.
60. Piccinini, G. *Physical Computation – A mechanistic Account*. Oxford University Press, 2015.
61. Pratt, V.R. Semantical considerations on Floyd-Hoare Logic. In *SFCS '76 Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, pp. 109-121, 1976.
62. Primiero, G. Realist Consequence, Epistemic Inference, Computational Correctness. In Arnold Koslow, Arthur Buchsbaum (eds.), *The Road to Universal Logic*, vol.2, pp. 573–588, 2015.
63. Primiero, G. Information in the Philosophy of Computer Science. In L. Floridi (ed.), *The Routledge Handbook of Philosophy of Information*, ch.10, pp. 90–106, 2016.
64. Rapaport, W. J. Implementation Is Semantic Interpretation. *The Monist*, 82(1): 109–30, 1999.
65. Rapaport, W. J. Implementation as Semantic Interpretation: Further Thoughts. *Journal of Experimental & Theoretical Artificial Intelligence*, 17(4): 385–417, 2005.
66. Schwitzgebel, E. Introspection. *The Stanford Encyclopedia of Philosophy* (Summer 2014 Edition), Edward N. Zalta (ed.), `http://plato.stanford.edu/archives/sum2014/entries/introspection/`.
67. Shoemaker, S. Self-Knowledge and 'Inner-Sense'. *Philosophy and Phenomenological Research*, 54:249–314, 1994. Reprinted in *The First Person Perspective and other Essays*, OUP, 1996.
68. Sonntag, D. On Introspection, Metacognitive Control and Augmented Data Mining Live Cycles. CoRR abs/0807.4417, 2008.
69. Sørensen, M. H., Urzyczyn, P. *Lectures on the Curry-Howard isomorphism*. Studies in Logic and the Foundations of Mathematics vol. 149, Elsevier Science, 2006.
70. Sotala, K., Yampolskiy, R. Responses to catastrophic AGI risk: a survey. *Physica Scripta*, 90, Royal Swedish Academy of Sciences, IOP, 2015.
71. Stalnaker, R. The Problem of Logical Omniscience, I. *Synthese*, 89:425–440, 1991.

72. Voss, P. Essentials of General Intelligence: The Direct Path to Artificial General Intelligence. In [37], pp. 131-158.
73. Yudkowsky E. *Value is fragile.* Less Wrong `http://lesswrong.com/lw/y3/value_is_fragile/`
74. Wiedermann, J. A Computability Argument against Superintelligence. *Cognitive Computation*, 4(3):236–245, 2012.
75. Wright, C. Self-Knowledge: The Wittgensteinean Legacy. in C. Wright, B. Smith, C. MacDonald (eds.), *Knowing our own Minds.* Oxford University Press, Oxford, 2000.