Minds & Machines (2014) 24:439–476
DOI 10.1007/s11023-014-9349-3

The Comprehensibility Theorem and the Foundations of Artificial Intelligence

Arthur Charlesworth

Received: 18 January 2014 / Accepted: 19 August 2014 / Published online: 6 November 2014

Abstract Problem-solving software that is not-necessarily infallible is central to AI. Such software whose correctness and incorrectness properties are deducible by agents is an issue at the foundations of AI. The Comprehensibility Theorem, which appeared in a journal for specialists in formal mathematical logic, might provide a limitation concerning this issue and might be applicable to any agents, regardless of whether the agents are artificial or natural. The present article, aimed at researchers interested in the foundations of AI, addresses many questions related to that theorem, including differences between it and results of Gödel and Turing that have sometimes played key roles in Minds and Machines articles. This study also suggests that—if one is willing to assume a thesis due to Donald Knuth—the Comprehensibility Theorem is the first mathematical theorem implying the impossibility of any AI agent or natural agent—including a not-necessarily infallible human agent—satisfying a rigorous and deductive interpretation of the self-comprehensibility challenge. Some have pointed out the difficulty of self-comprehensibility, even according to presumably a less rigorous interpretation. This includes Socrates, who considered it to be among the most important of intellectual tasks. Self-comprehensibility in some form might be essential for a kind of self-reflection useful for self-improvement that might enable some agents to increase their success. We use the methods of applied mathematics, rather than philosophy, although some topics considered could be of interest to philosophers.

Keywords Fallibility · Formal Peano Arithmetic · Formal proofs · Foundations of AI · Halting problems

Electronic supplementary material The online version of this article (doi:10.1007/s11023-014-9349-3) contains supplementary material, which is available to authorized users.

A. Charlesworth (✉)
Mathematics and Computer Science, University of Richmond, Richmond, VA 23173, USA
e-mail: ArthurCharlesworth@gmail.com

Introduction

Problem-solving software that is not-necessarily infallible is central to AI. Such software whose correctness and incorrectness properties are deducible by agents—both artificial and natural agents, and where the agents themselves are not-necessarily infallible—is an issue at the foundations of AI. A related theorem would help to define the boundaries of AI research related to agents and software, whether the software is written by humans or generated artificially. If such a theorem existed, having a working knowledge of it could be helpful for AI researchers, just as it is helpful for computer scientists to have a working knowledge of limitative results such as NP-completeness and the unsolvability of the halting problem.

The Comprehensibility Theorem, published in a journal for specialists in formal mathematical logic (Charlesworth 2006), appears to be such a theorem. The present article examines the extent to which this impression withstands additional study, addressing many questions about the theorem.

It is well-known that *arguments attempting to apply Gödel-Turing like results*—by that phrase we mean attempts to apply variants of Gödel's Theorem and/or the unsolvability of the halting problem to the processing of software by agents—have failed. For instance, this is made clear in an AI journal article (LaForte et al. 1998), whose conclusions include the following, *where we indicate in italics shortcomings of all arguments that have attempted to apply Gödel-Turing like results*:

- It is plausible that a generally-intelligent agent could be fallible, such as failing to be sound or even consistent. *Although AI researchers (see our section* "How Important is Eliminating Infallibility Assumptions About Intelligent Agents?") *agree with that conclusion, the attempts to apply Gödel-Turing like results fail to be applicable to a possibly-fallible agent's processing of software.*
- The attempts to apply Gödel-Turing like results use fallacious reasoning that can lead to incoherence similar to that of the liar paradox. *The attempts to apply Gödel-Turing like results provide no theorem, whose proof satisfies the current standard criterion for rigor, relating the ability to deduce correctness-properties of software to the ability to deduce an intelligence-related limitation of that software.*
- One fallacious property of such attempts to apply Gödel-Turing like results is their self-referent application—explicitly within an argument itself—of the intuitive concept "the method mathematicians use to correctly decide problems", and thus such arguments rely crucially on the fact that the agents considered are humans. *Even if they were rigorous, the attempts to apply Gödel-Turing like results would not provide a result, about the comprehensibility of software, that is applicable to non-human agents.*

This article studies in some depth the extent to which the Comprehensibility Theorem overcomes the above three italicized shortcomings of previous treatments.

The standard way to avoid the kind of incoherence similar to that of the liar paradox is via the use of a hierarchy, such as that obtained by using formal proofs as

objects within an appropriate mathematical model. It is natural for such formal proofs to be within systems that extend first-order Peano Arithmetic. The Comprehensibility Theorem appeared in a journal aimed specifically for an audience of specialists having deep expertise with such systems, and only brief explanation accompanied the construction of the underlying mathematical model, which one must understand to properly evaluate answers to questions about the model and the theorem. It is thus likely that those AI researchers without deep expertise in first-order Peano Arithmetic could encounter significant difficulty gaining an adequate knowledge of the theorem, and even some with such expertise might have difficulty. This article aims to meet the resulting need, before it addresses previously unaddressed questions.

Since commentaries like (LaForte et al. 1998) already exist that adequately discuss the details of, and cite references to, non-rigorous arguments attempting to apply Gödel-Turing like results, the present article avoids doing so. As mentioned by LaForte et al., it is well-known that such arguments rely on infallibility assumptions about agents. This article shows that one can view the Comprehensibility Theorem's proof as related to a non-rigorous argument sketched in our section "A Conjecture and a Nonrigorous "Proof"" that avoids such infallibility assumptions. [Analogously, the presentation of the Incompleteness Theorem (Gödel 1931) neither discusses details of nor cites references to the Richard paradox or the liar paradox, and instead explains that the proof of that theorem is related to a non-rigorous argument based on the phrase "this statement is not provable".]

The article is organized into sections as follows. "Brief Summary of the Comprehensibility Theorem" briefly summarizes the Comprehensibility Theorem. "Significance of Halting Problems" provides insight into the significance of halting problems, since the theorem relates to solving such problems. "A Treatment Avoiding Infallibility Assumptions, But Lacking Rigor" presents a non-rigorous argument providing intuitive motivation for the mathematical model. "The Mathematical Model" explains the construction of that model and "How the Comprehensibility Theorem Fits into the Model" looks at how the theorem fits into the model. "Applying the Comprehensibility Theorem to Real-World Agents" examines the application of the theorem in the real-world. "Questions About the Model and the Comprehensibility Theorem" addresses many questions about the theorem. (The author thinks those are likely the questions of broadest interest; the Electronic Supplement's sections address additional questions, cited within this article as ES1, ES2, etc.) "Conclusions" presents conclusions.

Each heading later in the article for the above sections appears in boldface. This article also contains over two-dozen sections whose headings do not appear in boldface. The Electronic Supplement contains an outline that includes the headings of *all* sections in this article.

Brief Summary of the Comprehensibility Theorem

The theorem is unlike theorems of Gödel and Turing. Rather than being about incompleteness or unsolvability, it is about the ability of an agent to make certain

deductions. The gist of the theorem is as follows, where $\mathcal{A}$ is an agent and S is software:

> *If $\mathcal{A}$ can correctly deduce correctness-related properties of S, then $\mathcal{A}$ can correctly deduce S does not have $\mathcal{A}$'s capability for solving halting problems.*

Roughly speaking, saying "$\mathcal{A}$ can correctly deduce correctness-related properties of S" means $\mathcal{A}$ can yield certain proofs in a formal system $\mathbb{F}$, concerning S's ability to solve halting problems, and saying "S does not have $\mathcal{A}$'s capability for solving halting problems" means there is a halting problem that $\mathcal{A}$ settles correctly and deductively but S does not settle correctly. The formal system $\mathbb{F}$ is required to be "adequate". Questions related to halting problems as well as attempts at formal proofs are coded as natural numbers, and an agent in this context is represented by a function $\mathcal{A}$ from a set of natural numbers to the set of natural numbers.

Although the proof of the theorem in Charlesworth (2006) is entirely about the ability to make deductions, that article uses the word "understand" in the statement of the theorem. To achieve greater clarity the present article avoids using that word in the theorem. (That word does appear in this article, related to quotes of Saunders Mac Lane, Donald Knuth, and Douglas Hofstadter, in our sections "Hilbert's Thesis", "Are Deductions About Scientific Phenomena Related to Deductions About Computer Programs?", and "Is Gödel's Second Incompleteness Theorem Mathematically Relevant to the Mind?".)

The above summary is vague. To avoid well-known pitfalls of paradoxical reasoning, crucially needed are *mathematical* definitions of such terminology as "agent", "correctly deduce", and "adequate". The next two sections provide what we hope is substantially more-helpful motivation for the definitions (and proof of the theorem), than that given by Charlesworth (2006). Those definitions then appear in "The Mathematical Model" section.

It is important to interpret technical results—such as about AI "belief" nets, "expert" systems, and "knowledge" bases investigated in other articles, and about "agent", "correctly deduce", "adequate", etc., in this article related to the Comprehensibility Theorem—using their accompanying definitions and constructions rather than other possible interpretations of the same words. It is also important to distinguish between the concepts of truth and provability; see the end of section "Gödel's Second Incompleteness Theorem is a Halting Problem Result".

Significance of Halting Problems

As indicated above, the theorem is specifically related to solving halting problems. How significant are halting problems? One can obtain insight by considering halting problems in mathematics and in logic.

Some Results and Conjectures that are Halting Problems

Perhaps the two most highly publicized mathematical results of the last half of the twentieth century were the proofs of the Four Color Theorem and Fermat's Last

Theorem. This section shows how a simple counterexample-seeking approach easily enables one to see that these two theorems concern halting problems.

The Four Color Theorem asserts that each flat ("planar") map is colorable using four colors so adjacent countries have different colors; equivalently, each planar graph is colorable using four colors so adjacent vertices have different colors. It is straightforward to construct software that generates the planar graphs using Kuratowski's theorem (van Lint and Wilson 1992), tries all possible combinations of a particular set of four colors on each so as to satisfy the stated constraint, and halts if and only if it finds a graph that cannot be so colored. The Four Color Theorem is equivalent to saying that the software just described fails to halt.

For Fermat's Last Theorem, the search program seeks an example of four positive integers a, b, c, and n, with $n > 2$, such that $a^n + b^n = c^n$, and uses a dovetailing approach in which the upper limit on the size of the four integers begins at 3 and is incremented after exhaustive search at each previous value of that limit. Such a program P_F can be less than two-dozen lines, written in a high-level programming language. Yet it took mathematicians over 350 years to show that there are no such positive integers (equivalently, that P_F fails to halt).

Someone limited to the demonstrated ability of Fermat, who in 1637 made his assertion that there are no such integers, would assert that P_F fails to halt, which is correct, assuming the proof of Fermat's Last Theorem by Andrew Wiles (1995) is correct. But that does not show that Fermat, who published no proof of his assertion, could "correctly deduce" his assertion, according to the customary meaning of that phrase. Part of the mathematical model is motivated by the fact that it was his proof that demonstrated Wiles made such a correct deduction.

One can use the same counterexample-seeking approach to see that some well-known unsolved problems—such as the conjecture that there are no odd perfect numbers and Goldbach's conjecture—are also halting problems. The point here is to gain insight into the significance of halting problems, rather than to suggest that counterexample-seeking programs provide an effective way to settle such conjectures. (See the end of section "Hilbert's Thesis Implies Each Rigorous Math Conjecture is a Halting Problem".)

It is not possible to use the simple counterexample-seeking approach of this section to view all mathematical conjectures as halting problems. Consider, for instance, the twin prime conjecture: there are infinitely many pairs of primes, such as 3 and 5, that are successive odd integers. (The simple approach does not suffice because the twin prime conjecture is not a Π_1^0 problem; see ES9.)

Gödel's Second Incompleteness Theorem is a Halting Problem Result

This section further explains the importance of halting problems, by indicating how one can view as a halting problem result Gödel's Second Incompleteness Theorem, which implies Gödel's Incompleteness Theorem and is thus perhaps the most significant theorem of logic. We begin with a brief review of Peano arithmetic, which is used throughout the rest of the article.

The reader is warned that we shall use the adjective "formal" as logicians use it, to mean "based solely on form". (In contrast, many technical articles use that

adjective as just a synonym for "mathematical", when giving what they call "a formal definition".) Peano arithmetic (PA) is a formal system that is first-order—its formal language has computer-checkable syntax rules and permits just a single variable type—whose axioms express relationships among logical symbols, including $=$, as well as among the symbols 0, $+$, s, $\times$, and $<$. For instance, an axiom of PA is $\forall x(x + 0 = x)$. A **formal proof** in PA is a finite list of sentences of PA each of which either is an axiom of PA or follows via a rule of inference of PA from preceding sentences in the list. A **formal theorem** of PA is a sentence that occurs as the last member of at least one formal proof in PA.

The (non "formal") **standard interpretation**, $\mathcal{N}$, of PA is the interpretation of the formal language in which each variable represents a non-negative integer, 0 represents zero, $+$ represents addition, s represents successor ($s0$ represents one, $ss0$ represents two, etc.), $\times$ represents multiplication, $<$ represents less-than, and the logical symbols also have their usual interpretations, for instance, $\neg$ represents negation. To illustrate this, when the axiom $\forall x(x + 0 = x)$ is interpreted via $\mathcal{N}$ and then expressed in English, the result is the assertion that the sum of any non-negative integer and zero is equal to the non-negative integer itself.

This article assumes all axioms of PA are true (according to $\mathcal{N}$) and all rules of inference of PA preserve truth, and hence that PA is **sound**; that is, all the formal theorems of PA are true. As explained in "How Can One Assume that a System Used by an Agent is Sound Without Assuming Agent Infallibility?", this widely-accepted soundness assumption is altogether different from assumptions about the infallibility of agents. To say PA is **consistent** means there is no sentence A in the language of PA such that both A and $\neg A$ are formal theorems of PA. It follows from the soundness of PA that PA is consistent.

The set of formal theorems of PA is computably enumerable. This follows from a well-known dovetailing argument based on two facts: (1) there is an algorithm to generate the axioms of PA and (2) for each rule of inference of PA there is an algorithm to carry out the effect of that rule. Thus, there is a Turing machine S that takes as input the code for any sentence A of PA and exhaustively seeks a formal proof of A, halting if and only if it finds such a formal proof. Gödel proved there is a sentence Con_{PA} in PA whose meaning under $\mathcal{N}$ is that PA is consistent, whose truth we assume as explained above. When applied to PA, Gödel's Second Incompleteness Theorem states that Con_{PA} is not provable in PA. That is equivalent to saying: S, when given as input the code for Con_{PA}, fails to halt. It is this sense in which one can view Gödel's Second Theorem as a halting problem result.

The preceding paragraph also provides an example that helps us emphasize the *difference between truth and the concept of provability in a formal system*: Assuming the consistency of PA, Con_{PA} is true according to $\mathcal{N}$ but, by Gödel's Second Incompleteness Theorem, Con_{PA} is not provable in PA. The Comprehensibility Theorem is about the concept of provability.

Hilbert's Thesis

The current standard criterion for a proof to be considered mathematically "rigorous" is that it can be carried out in principle within the constraints of Zermelo-Fraenkel set theory, abbreviated ZF. Many working mathematicians extend this to include the Axiom of Choice, to get the system ZFC, whose consistency is implied by the consistency of ZF, according to a theorem of Gödel. (The distinction between ZF and ZFC is not important in this article.) In the words of Saunders Mac Lane:

> Proof in Mathematics is both a means to understand why some result holds and a way to achieve precision. As to precision, we have now stated an absolute standard of rigor: A Mathematical proof is rigorous when it is (or could be) written out in the first-order predicate language $L(\in)$ as a sequence of inferences from the axioms ZFC, each inference made according to one of the stated rules. (Mac Lane 1986 p. 377)

Hilbert's Thesis is the assertion that *each provable mathematical statement can be expressed in the first-order language of set theory and has a formal proof in ZF (or in ZFC)*; for example, see (Hodel 1995 p. 263), which also observes:

> Isolating the axioms of Zermelo-Frankel set theory is a remarkable and outstanding achievement. The formal system ZFC is a clear, precise answer to this question: What logical axioms, rules of inference and nonlogical axioms suffice to derive the theorems of modern mathematics?

The typical mathematician assumes Hilbert's Thesis as a working hypothesis, pursuing mathematical results while accepting the current standard criterion for rigor as the requirement on all rigorous proofs, and also assumes as a working hypothesis that ZF is consistent. The typical mathematician is not actually concerned about the rigorous formal foundations of that discipline. But it is possible for one to view the research of the typical mathematician as a process that proceeds by adopting the working hypothesis that ZF is consistent and then seeks proofs of conjectures, where the proofs are formalizable in principle within ZF. Of course, a proof of a counterexample is itself a proof of a conjecture.

It is important here to mention that, although the proof of the Comprehensibility Theorem itself is formalizable in principle in ZF, it does not assume that all proofs must be formalizable in ZF. This article aims to be as broadly applicable and as independent of points-of-view as feasible: We do not assert that assuming Hilbert's Thesis is the only reasonable approach, that the only reasonable formalization of a problem must be within ZF, or even that all acceptable mathematics must be formalizable in principle within a system whose formal theorems are computably enumerable; see ES9.

Hilbert's Thesis Implies Each Rigorous Math Conjecture is a Halting Problem

It is easy to see that the formal theorems of ZF, like those of PA, are computably enumerable. Let S_{ZF} denote a Turing machine that takes as input the code for any

sentence A of ZF and exhaustively seeks a proof of A in ZF, halting if and only if it finds such a formal proof. The construction of S_{ZF} is like that of S in "Gödel's Second Incompleteness Theorem is a Halting Problem Result", except use the axioms and rules of inference of ZF instead of those of PA.

To further indicate the importance of halting problems, this paragraph shows that it follows from Hilbert's Thesis that *each rigorous mathematical conjecture is a halting problem*. Assume Hilbert's Thesis. Let c be any conjecture in mathematics, and let A_c be a sentence within ZF that expresses c. The conjecture c can be any already-settled result, such as Fermat's Last Theorem; the conjecture c can also be any unsettled claim, such as the twin prime conjecture, the conjecture that $P \neq NP$, and any probabilistic conjecture expressible in ZF made by a robot about a possible consequence of that robot's current assumptions, plans, and percepts. The problem of proving c in ZF is equivalent to the halting problem: does S_{ZF} halt when given as input the code for A_c? That is, a "rigorous" proof of the "yes" answer to the just-stated halting problem is equivalent to a "rigorous" proof of c. (Also note that for any particular c one can embed A_c within S_{ZF} to obtain a Turing machine P_c *with a blank input tape* whose halting problem yes-no answer, if affirmative, is equivalent to proving c.)

Continue to assume Hilbert's Thesis through the current paragraph. It is possible that neither A_c nor its negation can be proven within ZF; that is, that A_c is "undecidable" in ZF. Even that possibility itself is also equivalent to a halting problem. To see this, consider the modification of S_{ZF} that halts if and only if it finds a formal proof either of A_c or of $\neg A_c$. An example of such an undecidable sentence in ZF—and hence an undecidable conjecture in rigorous mathematics—is the Continuum Hypothesis CH, which asserts that for every infinite subset of the set of real numbers there exists either a one-to-one correspondence between that subset and the set of natural numbers or else there exists a one-to-one correspondence between that subset and the set of real numbers.

Princeton professor John J. Burgess has written:

> The mathematical community as a whole has … abandoned any attempts to solve by mainstream methods a question mathematical logicians have inferred to be undecidable. The work of the few who are exceptions to this generalization bears all the signs of being the work of crackpots. (Burgess 1992)

The present author prefers avoiding the word "crackpots" to describe people, but the above quote helps indicate the very strong acceptance of Hilbert's Thesis. (As mentioned at the end of the preceding section, the current article aims to be as broadly applicable as possible, and does not assume Hilbert's Thesis is the only reasonable approach.)

Before leaving this discussion of Hilbert's Thesis, we should point out that this article does not confuse the formalization of a conjecture with the formalization of the provability of a conjecture; see ES9.

We also point out that there are many important historical examples where an effective framework into which to translate a mathematical problem is quite different from that of the initial problem itself. For instance, algebraic geometry is

used in the proof of Fermat's Last Theorem by Wiles (1995), which solves a halting problem, even in the non-formal sense of section "Some Results and Conjectures That Are Halting Problems". Thus, investigating the software P_c directly need not be an effective way to settle the conjecture c, and an exhaustive search for a proof in ZF seems an unreasonable way to prove a result, even though the latter is guaranteed to succeed any time such a proof exists, assuming one could carry out a huge number of steps flawlessly (an assumption avoided in this article).

A Game Based on Solving Halting Problems

The most fundamental of concepts are those arising in a variety of contexts. That is the case for halting problems. As just explained, Hilbert's Thesis—assumed as a working hypothesis by nearly all mathematicians—implies that each rigorous mathematical conjecture is equivalent to a halting problem. Without using formal logic, "Some Results and Conjectures That Are Halting Problems" explains how it is easy for one to view as halting problems two highly publicized mathematical results of the latter half of the twentieth century. Douglas Hofstadter (1982) asserts that "the ability to break out of loops of all types seems the antithesis of the mechanical". Roboticist Hans Moravec (1992) speculates on the usefulness of giving a robot a primitive consciousness-like ability via a heuristic infinite loop detector: "If you have a maze-running robot, if it has any way of discovering that it's just repeating its actions, then you have some way of getting out of that loop." Moreover, solving halting problems can enhance the success of commonly used operating systems and other practical software; a team coordinated by Microsoft Research, Cambridge, has recently developed software that discovers halting proofs for some practical operating system programs (Cook et al. 2006) and, in the opposite direction, another team has recently developed software that discovers non-halting proofs for some practical programs including a permissions-controller for memory-protection in an operating system (Gupta et al. 2008). (See ES3 and ES4; the proofs mentioned in the preceding sentence are not formal proofs, but see "Is it Necessary to Expect an Agent to Give Formal Proofs?".)

Due to the fundamental nature of halting problems, it is reasonable when constructing a mathematical model related to intelligence to consider a facet of intelligence related to solving them. Intuitively speaking, if an agent possesses general intelligence, then the agent should perform well in this facet of intelligence; of course the converse need not hold. The mathematical model underlying the Comprehensibility Theorem is motivated by focusing on the fundamental capabilities of agents, rather than the relative speeds of agents.

The game in Fig. 1 conveys some intuitive ideas behind the model. Let us call the game CHESS, an acronym for CHEcking the Stopping of Software, using all capital letters to distinguish it from the board game of chess. The tie-breaker rule in the last sentence of that figure can be used when one player does not respond after a long period of time; a proof that that player will not make the correct decision is conceivable, for instance, if that player is software that the other player can examine. The CHESS game requires no infallibility assumption on the players: a player can make an incorrect decision and a player can also give a proof attempt that is incorrect. Whether a player has indeed won an instance of CHESS depends on the correctness of

Each of two players receives the text of a program P that contains no input statements. A player can be software, robotic, human, or any other kind of agent. Each player tries to decide without external help whether or not P halts when run, assuming unlimited time and computer memory are available. The possible responses by the players are:

"Input halts"	if the player has decided that P halts
"Input does not halt"	if the player has decided that P does not halt
no response	if the player has not decided whether or not P halts

A player is permitted to make at most one decision corresponding to the input P. One way a player can win the game involving P is to make the correct decision and for the other player to make the incorrect decision. The game has no time limits, but a player can also win by making the correct decision, justifying that decision with a (correct) proof of its correctness, and also giving a (correct) proof that the other player will not make the correct decision.

Fig. 1 Rules for the game of CHEcking the stopping of software

decisions and sometimes also on the correctness of a proof attempt by a player. Perfect judgments about correctness may lie beyond human capabilities; the mathematical model described in "The Mathematical Model" is independent of that issue.

This game is mentioned briefly—as the "halting problem game"—in Charlesworth (2006), but that article lacks the motivation for the proof of the Comprehensibility Theorem provided by the non-rigorous argument in our next section.

A Treatment Avoiding Infallibility Assumptions, but Lacking Rigor

This section presents a nonrigorous argument related to a particular conjecture. We then explain why the treatment in this section lacks rigor and *does not yield any theorem*.

A Conjecture and a Nonrigorous "Proof"

The purpose of this section is two-fold: to explain some intuitive ideas related to the Comprehensibility Theorem, and to provide an opportunity for pointing out the need for much greater rigor in this context. The section assumes that one of the two players in the CHESS game of Fig. 1 is human in nature, but the mathematical model presented after this section adopts a much more general viewpoint. The phrase "actual human" appearing in the following conjecture emphasizes the lack of an assumption that humans are infallible.

> Conjecture: *If the set A of all actual humans can correctly deduce correctness-related properties of software S, then A can correctly deduce S does not have A 's capability in the CHESS game.*

Saying a player A "can correctly deduce correctness-related properties" of software S (whose output, if any, is an attempt to settle the halting problem for the corresponding input), intuitively means that, given the source code for S:

1. For each specific input for S: A can correctly deduce whether or not S makes an incorrect decision and A can provide a correct proof of that deduction.

2. For each specific input for *S*: if *S* produces output, then *A* can correctly deduce that output and *A* can provide a correct proof of that deduction.
3. *A* can correctly deduce (and correctly justify its deduction) that for each input of *S* that yields output, that output attempts to settle the halting problem for the corresponding input.

Let us refer to this three-part assumption as the "correct deduction assumption".

By saying a player *A* can "correctly deduce that *S* does not have *A*'s capability in the CHESS game" we intuitively mean that there exists an instance of the CHESS game such that *A* can correctly deduce that *S* loses to *A* in that instance of the game.

The rest of this section presents a "proof" of the Conjecture that is a modification of the proof of the unsolvability of the halting problem. For simplicity, we modify a concise proof of that result based on the use of a pointer to a subprogram (Dorin and Toal 1995). The Ada programming language is used here, since it provides a high-level of support for pointers to subprograms, via its **access** reserved word.

Suppose *A* and *S* satisfy the hypothesis of the Conjecture. We show that *S* loses to *A* in the instance of CHESS played with a particular computer program that we construct.

We assume without loss of generality that, for any input program *P*, procedure *S* either generates no output or prints exactly one of "Input halts" and "Input does not halt" and:

prints "Input halts" and halts	if *S* decides that *P* halts
prints "Input does not halt" and halts	if *S* decides that *P* does not halt
no response	if *S* cannot decide whether or not *P* halts

This *does not imply that all—or even some particular small fraction—of S 's decisions about the halting of programs are correct*, which sharply distinguishes the approach of this game from the approach of standard Gödel-Turing arguments. We assume without loss of generality that the final statement in *S* is **return**; that the first of the above three actions results from executing Put("Input halts"); **return**; the second results from executing Put("Input does not halt"); **return**; and the third either results from executing a return statement without having generated output or from *S*'s own failure to halt. Define a function Check with declarations

```
type AccessToProcedure is access procedure;
function Check (P: AccessToProcedure) return Boolean;
```

and so that Check(P'access) checks what happens when *P* is executed, according to *S*, and:

returns True	if *S* decides that *P* halts
returns False	if *S* decides that *P* does not halt
does not halt	if *S* cannot decide whether or not *P* halts

To construct Check from *S*, first change a procedure into a function, and then replace each occurrence of Put("Input halts"); **return**; in *S* by **return**(True); replace each occurrence of Put("Input does not halt"); **return**; by **return**(False); and replace each remaining return statement by the explicit infinite loop **loop null**; **end loop**;

Now define the procedure CheckMate in Fig. 2. We complete the "proof" by "proving" the claim *CheckMate is an instance of CHESS that S loses when playing A*.

We first consider the case in which *S* prints an incorrect output when presented with CheckMate. In this case, the correct deduction assumption implies it is possible for *A* to both correctly deduce that fact and to correctly deduce what *S*'s output is (as well as to correctly prove those deductions). Knowing which of the two outputs is incorrect enables *A* to deduce the other output is correct. The italicized claim in the preceding paragraph thus follows in this case.

Thus we can focus in the rest of the "proof" on the case in which *S* either prints the correct output or gives no response when presented with the data CheckMate.

Because of the way Check is constructed from *S*, Check(CheckMate'access) either returns the correct boolean value or does not halt. Thus Check(CheckMate'access) does not return True, since that would imply that CheckMate halts and also, due to the way CheckMate is defined, directly cause CheckMate not to halt. It is also the case that Check(CheckMate'access) does not return False, since that would imply that CheckMate does not halt and directly cause CheckMate to halt. Thus Check(CheckMate'access) cannot halt, so the definition of CheckMate makes it clear that CheckMate itself cannot halt and thus the correct decision in the instance CheckMate of CHESS is "Input does not halt". Because of the way Check is constructed from *S*, it is also clear that *S* would give no response in the instance CheckMate of CHESS.

It is possible for the set *A* of humans to correctly deduce the facts in the preceding paragraph (since we have just done so), so the italicized claim follows.

The Inherent Lack of Rigor in the Preceding "Proof"

The end of the above "proof" depends on the participation of humans *inside* the "proof". We clarify this point by indicating the sense in which it is possible for *A* to construct its proof attempt in the game.

There are three situations *A* could confront.

1. *S prints "Input halts", given input CheckMate, and that is an incorrect decision.* By the correct deduction assumption, it is possible for *A* to write a correct proof of that deduction. It is then possible for *A* to follow up that correct proof with the decision "Input does not halt", which is correct.
2. *S prints "Input does not halt", given input CheckMate, and that is an incorrect decision.* (Is similar to 1.)
3. *S does not print incorrect output, given input CheckMate.* By the correct deduction assumption, it is possible for *A* to give a correct proof of exactly that

```
procedure CheckMate is
begin
    if Check(CheckMate'access) then
        -- If S decides CheckMate halts, then CheckMate does not halt.
        loop null; end loop;
    end if;
    -- If S decides CheckMate does not halt, then CheckMate halts.
end CheckMate;
```

Fig. 2 Program used in "proof" of Conjecture

fact. It is then possible for *A* to extend that proof with the next to last paragraph of section "A Conjecture and a Nonrigorous "Proof"", to obtain the required proof mentioned in the tie-breaking rule of Fig. 1.

The "proof" of the Conjecture lacks rigor for these reasons:

- The Conjecture and its "proof" both depend crucially on the concept of "human", yet there is no known way to define that concept rigorously; that is, so that in principle the definition can be given in ZF.
- The statement of the Conjecture and its "proof" both depend on an undefined concept of a "correct proof".
- The final sentence of the "proof" makes an inherently nonrigorous use of self-reference in depending on the participation of humans *inside* the "proof". That is, the "proof" assumes and uses undefined capabilities of human reasoning.

The risk of using undefined capabilities of human reasoning within an argument is easy to see. Here is a concise example of a fallacious "proof" in the context of human deductions, based on the vague concept of "correctly prove" and the nonrigorous use of self-reference in which humans participate inside the "proof". Let *S* be the sentence "No human can correctly prove that this sentence is true", so that the meaning of *S* is "No human can correctly prove that *S* is true".

Claim *S* is a true sentence and no human can correctly prove that *S* is true, demonstrating a limitation to correct human deductions.

Proof: Because of the meaning of *S*, to prove the Claim it suffices to show that *S* is true. But that is clear, since if *S* were false, then "Some human can correctly prove that *S* is true" would hold, which would imply that *S* is true and that would result in a contradiction. □

It is easy to see that this "proof" uses fallacious reasoning. If the "proof" supporting the Claim is correct, then the Claim is true and, in addition, some human (namely the writer) correctly proves that *S* is true, from which it follows that the Claim is false, a contradiction.

There are strong similarities between the "proof" of the Claim in the preceding paragraph and the "proof" given in the preceding section. Attempts to apply variants of Gödel's Theorem and/or the unsolvability of the halting problem to an

agent's ability to deduce properties of software have failed similarly, due to the logical fallacy in

> the idea that it is actually meaningful to talk about "the method mathematicians use to correctly decide problems". Like "truth", this phrase simply cannot be given an exact meaning which mirrors in every way its informal use, and assuming it can leads one directly into the kind of incoherence familiar from the liar paradox (LaForte et al. 1998 pp. 276–277).

Obtaining a proof of a result like the Comprehensibility Theorem, regarding the ability of agents to make certain deductions about software, requires constructing an acceptable *mathematical* model. Even if the goal is to convey just the central ideas to a broad audience, it is not possible to sketch the above intuitive idea behind such a proof and leave it at that, since that intuitive idea relies on fallacious logic.

The Mathematical Model

As we have seen, the "proof" in "A Conjecture and a Nonrigorous "Proof"" uses a self-reference argument similar to an argument that produces paradoxical nonsense. The purpose of that section is to provide intuitive motivation for the mathematical model presented in this section. This section reviews Turing machines and explains conventions used in this article. The section then presents a lemma that permits the model to avoid strong infallibility assumptions on agents, and describes how that lemma leads to a particular set of true formulas in Peano arithmetic, the formal system reviewed in "Gödel's Second Incompleteness Theorem is a Halting Problem Result". The section then explains how that set of formulas is used in the definition of "correctly deduce correctness-related properties" in a way that permits the proof of the Comprehensibility Theorem to avoid the kind of dangerously non-rigorous reasoning identified in "The Inherent Lack of Rigor in the Preceding "Proof"".

Turing Machines and Codes

A standard methodology for avoiding paradoxical nonsense in arguments that involve self-reference is to provide fully mathematical definitions of all concepts, and to use a hierarchy of languages. That is the methodology used within the mathematical model. For instance, it is essential that the model require any "proof" given by a player in the CHESS game of Fig. 1 to be a mathematically-defined (rather than intuitively-defined) object, namely a formal proof within a formal axiomatic system. Also, since there is no known rigorous definition of "human", of necessity the theorem is presented at a higher level of abstraction, so that the theorem is applicable to agents of any kind.

The S mentioned in the statement of the Comprehensibility Theorem is a Turing machine. That is appropriate since the Turing machine concept has a simple mathematical definition, yet captures the input/output capability of partial recursive functions of natural numbers. As a result, the Turing machine concept captures the full input/output functional capability—for natural numbers, which as pointed out in

our next paragraph is no fundamental restriction—of any current concept of software. The domain, $\mathcal{D}(P)$, of Turing machine P is the set of natural numbers n such that P halts when executed using n as its input; that is, when the contents of the initial tape for P is the unary code for n and the read head of P is over the leftmost position in that unary code. Corresponding to each Turing machine P is a natural number code $\#P$, such that the description of P can be computationally determined by decoding $\#P$.

Numerical codes permit the use of formulas of PA whose free variables can be bound to numerals for numbers, and which can thereby represent the formalizations of halting-related problems; the **numeral** in PA for a number is the formal notation for that number. Such codes also permit us to define an agent as simply a function from natural numbers to natural numbers, which does not restrict applications of the model, since any discretized input or output of a human, robot, or other system is a finite string of bits and one can view each finite string of bits as representing a natural number. This article—unlike (Charlesworth 2006)—eliminates nearly all explicit notation for the codes of Turing machines and formulas, via the use of simplified expository conventions explained in the next section. [It is possible to give a solid proof of a general version of Gödel's Theorem that entirely avoids numerical coding. Of necessity, numerical coding plays a more prevalent role in the mathematical model supporting the Comprehensibility Theorem.]

Conventions

In the halting-problem CHESS game of Fig. 1, each player is given the text of a program P that contains no input statements. The mathematical model of this game uses a Turing machine, rather than a program, for P. Also, rather than having a blank input tape, the Turing machine P has an input tape containing a specific pattern—denoted as $\langle P, Halt? \rangle$—which can be computationally decoded to yield P as well as an indication that the relevant problem is that of determining whether or not P halts with input $\langle P, Halt? \rangle$. The model supports presenting to an agent five different kinds of halting-related problems to solve. Let us use the phrase "*the* halting problem for P" to mean the problem of determining whether or not P halts with input $\langle P, Halt? \rangle$. There are many ways the coding of $\langle P, Halt? \rangle$ could have been achieved, so the choice is relatively unimportant.[1]

As explained at the beginning of "Turing Machines and Codes", any correct proof provided by a player in the CHESS game must be modeled as a formal proof. In particular, the model requires such formal proofs to be given within a system $\mathbb{F}$ that extends PA and satisfies additional properties described in an upcoming section that defines the concept of an "adequate" formal system. Halting problems are representable within PA, as now explained. It follows from standard techniques of logic (Charlesworth 2006 Lemma 7.1) that there is a formula of PA—denoted by *HLT*—with two free variables and these properties:

[1] For specificity, one can choose to let $\langle P, Halt? \rangle$ denote the unary code for the number $2^{\#P}3^1$, which is the prime-power code for the ordered-pair $(\#P, Halt?)$, where *Halt*? denotes the constant 1. Additional kinds of halting-related problems are supported similarly, using constants 2 through 5, instead of 1.

1. S with input $\langle P, Halt?\rangle$ halts iff $HLT(S,P)$ is true according to $\mathcal{N}$
   and
2. S with input $\langle P, Halt?\rangle$ halts iff $HLT(S,P)$ is a formal theorem of PA,

where $HLT(S,P)$ denotes the sentence of PA obtained by substituting the numerals for the codes of Turing machines S and P into the slots in HLT where the free occurrences of the two respective free variables occur. For simplicity, unlike Charlesworth (2006) this article uses notation that suppresses explicit use of numerical codes: notation such as $F(\cdots P \cdots)$, where F represents a formula in a formal system and P represents a Turing machine, will denote substituting the numeral for the numerical code for P into the corresponding slots in formula F where the free occurrences of the respective free variable occur.

For convenience, whenever this article refers to the meaning of a formula of PA, we assume the use of the standard interpretation $\mathcal{N}$ of the language of PA (see "Gödel's Second Incompleteness Theorem is a Halting Problem Result"). Also, the more concise notation $H(P)$ is used for $HLT(P,P)$, and for any Turing machine P the phrase "P halts" is used as an abbreviation for the phrase "P with input $\langle P, Halt?\rangle$ halts". For instance, it follows from these conventions and (1) that the meaning of the formula $H(P)$ is that P halts, and the meaning of $\neg H(P)$ is that P does not halt, where $\neg$ denotes negation.

Two requirements on a player that attempts to break a tie in an instance of the CHESS game are that the player make the correct decision in that instance of the game and that the player justify that particular decision with a proof of its correctness. To avoid one kind of infallibility assumption on agents, the model binds that player's attempted proof with the decision made by that player. This binding of proof with decision simplifies the model, by removing the otherwise possible situation—since infallibility assumptions are to be avoided—that the player makes one decision and yet that player's attempted proof of correctness of that decision is actually an attempt to prove something different, perhaps even the opposite decision!

Let $\mathcal{A}$ be a function from a subset of $\mathbb{N}$ to $\mathbb{N}$. The function $\mathcal{A}$ models an **agent** for answering halting problem related questions, in the following way. The value of $\mathcal{A}$ at $\langle P, Halt?\rangle$ is a **decision about** $\langle P, Halt?\rangle$ iff $\mathcal{A}(\langle P, Halt?\rangle)$ is a number that (a) can be computationally decoded in one way to produce exactly one of $H(P)$ and $\neg H(P)$, for which let us say $\mathcal{A}$ with input $\langle P, Halt?\rangle$ **decides that P halts** (respectively, **does not halt)** and (b) can be computationally decoded in another way to produce a list of codes of formulas, in a situation where $\mathcal{A}$ gives an attempted proof of the correctness of that decision; if such an attempted proof is indeed a (correct) formal proof in $\mathbb{F}$, then let us say that $\mathcal{A}(\langle P, Halt?\rangle)$ is **logically correct** in $\mathbb{F}$. The model supports this two-fold computational decoding in a straightforward way; the details are relatively unimportant.[2] The function $\mathcal{A}$ is a **decision system** iff for each Turing machine P:

[2] In order to be a decision about $\langle P, Halt?\rangle$, the model requires $\mathcal{A}(\langle P, Halt?\rangle)$ to be the code of a list of natural numbers of finite length (the length can be 1, such as when $\mathcal{A}$ does not attempt a proof) such that the last member of that list is the code either of $H(P)$ or of $\neg H(P)$, and $\mathcal{A}$'s proof attempt is logically correct in $\mathbb{F}$ iff $\mathcal{A}(\langle P, Halt?\rangle)$ is the code of a formal proof of $\mathbb{F}$. To avoid a circular definition of "adequate", the construction of the model does not use the part of this definition depending on the

$\langle P, Halt?\rangle \in \mathcal{D}(\mathcal{A})$ implies $[\mathcal{A}(\langle P, Halt?\rangle)$ is a decision about $\langle P, Halt?\rangle]$.

A Turing machine $\mathcal{A}$ is a **TM decision system** iff $\mathcal{A}$ is a decision system. [The author believes there is no important mathematical difference whatsoever between the proofs in Charlesworth (2006) and the proofs in this article, although the choice of terminology is different here. Instead of defining "decision system", that article defined "deductive system", in a way that would have the effect of requiring a formal proof by a player regardless of how the player would win the CHESS game introduced in this article. Also, for greater clarity, this article's statement of theorem and proof uses phrases involving "deduce" rather than phrases involving "understand".]

Rigorous Analogues for Both Check and CheckMate

A lemma guarantees a computational way to obtain from any TM decision system P a Turing machine P' such that, roughly speaking, the following holds. If P halts when asked to determine whether or not P' halts, then P's decision about that halting problem is incorrect; furthermore, if P goes into an infinite loop when asked to determine whether or not P' halts, then P' itself does not halt. The relationship between P and P' is analogous to the relationship between Check and CheckMate in the inherently-flawed "proof" of the Conjecture in "A Conjecture and a Nonrigorous "Proof"".

The definition of "decides" in our "Conventions" supports the following statement of the lemma, which is simpler than that in 7.15 of Charlesworth (2006):

Lemma *Let* $D = \{\#P | P$ *is a TM decision system*$\}$. *There exists a Turing machine* G *so that* $D \subseteq \mathcal{D}(G)$ *and for each TM decision system* P, *the number* $G(\#P)$ *encodes a TM decision system* P' *such that*:

(a) *if* P *with input* $\langle P', Halt?\rangle$ *decides that* P' *halts, then* P' *does not halt*
(b) *if* P *with input* $\langle P', Halt?\rangle$ *decides that* P' *does not halt, then* P' *halts*
(c) *if* $\langle P', Halt?\rangle \notin \mathcal{D}(P)$, *then* P' *does not halt*.

Like standard proofs of the unsolvability of the halting problem, the proof of the lemma uses a diagonal argument. Unlike the lemma, standard proofs of the unsolvability of the halting problem conclude with a contradiction having a form such as: P' halts iff P' does not halt. Also, whereas such standard proofs assume that 100 % of the decisions about halting questions made by a particular Turing machine P are correct, *the proof of the lemma does not assume that even 1 % of* P *'s decisions about halting questions are correct*. At first glance, P' might seem analogous to a Gödel sentence (a true sentence in a consistent formal axiomatic

Footnote 2 continued
particular system $\mathbb{F}$ until after a set of formulas $\mathcal{T}$—that do *not* depend on $\mathbb{F}$—is defined in "A Set of Formulas Supporting a Technical Trick". Using a list of natural numbers as described in this footnote is analogous to late binding within the implementation of dynamic functions in object-oriented languages (Appel 1998 Sect. 14.2).

system that is not provable in that system) in attempts to apply Gödel-like results to the comprehensibility of software by an agent, but such an analogy overlooks the above italicized point. Such attempts are based on investigating the consequence of assuming that an agent reasons using (correct) formal proofs and that the agent's reasoning system itself is *consistent*. Thus, such attempts assume the agent's reasoning is infallible in a strong way, since they assume the agent is *never* inconsistent. (See ES6.)

To see how the lemma avoids infallibility assumptions, notice that for each of parts (a) and (b) there exists a highly-fallible Turing machine that satisfies the hypothesis of that part; in fact, that makes infinitely many incorrect decisions about halting questions. A simple such P for which the hypothesis of (a) holds is the Turing machine that, for each Turing machine Q, takes input $\langle Q, Halt? \rangle$ and produces as output the code for the list whose only member is the code for $H(Q)$. Changing H to $\neg H$ in such an example yields a simple such P for which the hypothesis of (b) holds. It is also easy to give an example for which the hypothesis of (c) holds: just consider a Turing machine P that, for each Turing machine Q, takes input $\langle Q, Halt? \rangle$ and fails to halt.

The rest of the article uses the prime notation on a TM decision system, and applies the properties of that notation given by the lemma.

Arithmetization

As explained in our "Conventions", halting problems are formalizable in PA. Our section "A Set of Formulas Supporting a Technical Trick" will explain how additional aspects of the halting-problem CHESS game of Fig. 1 are also formalizable in PA. Many AI researchers are more familiar with other technical mathematics than with formal Peano Arithmetic. Thus, prior to explaining the model's additional use of formal logic shortly, we provide a well-known analogy. Ancient geometers might find surprising the investigation of geometrical figures since the time of Descartes, due to its use of algebraic expressions. The use of such analytic geometry is called the "arithmetization of geometry", because a key step of analytic geometry is translating geometrical properties into real-number arithmetical properties.

A seminal innovation of Gödel was the analogous arithmetization of metamathematics, in which properties relating to formal systems are translated into natural-number properties that, in turn, are formalizable in systems such as PA. This permitted Gödel to treat mathematically the idea expressed by the assertion "this statement is not provable", using methodology avoiding the non-rigorous reasoning that results from examining the truth of the assertion "this statement is false". That enabled Gödel to prove his Incompleteness Theorem in an acceptable mathematical way.

Similarly, this article's approach—unlike that of Charlesworth (2006)—explains how the dangerously non-rigorous treatment of the halting-problem CHESS game in "A Treatment Avoiding Infallibility Assumptions, But Lacking Rigor" can be replaced by a mathematically acceptable treatment. The next section continues to introduce notation about formulas in PA that represent concepts in the CHESS

game. A helpful perspective is to view this as analogous to the way algebraic expressions are used elsewhere to represent concepts in geometry.

Our exposition will not require readers to master the next section's details; whenever we use the notation of that section—such as in our section "Definition of "Correctly Deduce""—we accompany its use with a corresponding intuitive explanation. To have replaced all mention of the notation in the next section with sketchy exposition would have resulted in a vague and thus inadequate explanation of the model.

A Set of Formulas Supporting a Technical Trick

A technical trick enables the proof of the Comprehensibility Theorem in Charlesworth (2006) to avoid the non-rigorous reasoning we sketched in our section "The Inherent Lack of Rigor in the Preceding "Proof"". The trick uses a set $\mathcal{T}$ of true formulas, which we explain shortly. Upcoming sections show how $\mathcal{T}$ supports mathematical definitions of an "adequate" formal system, and of a "correctly deduce" concept like that mentioned in the statement of the nonrigorous Conjecture in our section "A Conjecture and a Nonrigorous "Proof"".

Recalling the game in that Conjecture, the names D, B, and R can suggest "decision system", "break-tie", and "result", respectively. The notation C can suggest the "partial correctness" terminology used in formal verification, which means "not producing incorrect output". There are four formulas of PA denoted as just mentioned such that these properties hold (Charlesworth 2006 Lemma 8.1):

- For each Turing machine P, the formula $D(P)$ is true iff P is a TM decision system. (Recall from our "Conventions" that notation such as $D(P)$ suppresses the explicit use of codes in this article.)
- For each TM decision system P and each Turing machine Q, the formula $B(P, Q)$ is true iff the following holds: If P with input $\langle Q, Halt?\rangle$ halts, then what P decides about the halting problem for Q is false. (Recall that "decides" is defined mathematically in our "Conventions".)
- For all Turing machines P and Q and for each formula A in PA, the formula $R(P, Q, A)$ is true iff the following holds: If P with input $\langle Q, Halt?\rangle$ halts, then the output is the code of a list of natural numbers whose last member is the code for A.
- For all Turing machines P and Q and each formula A in PA, if $H(P, Q) \wedge R(P, Q, A)$ is true, then $R(P, Q, A)$ is a formal theorem of PA. Here $\wedge$ denotes the logical "and" operation.
- For each TM decision system P and each Turing machine Q, the formula $C(P, Q)$ is true iff the following holds: If P with input $\langle Q, Halt?\rangle$ halts, then what P decides about the halting problem for Q is true.

Formulas D, B, R, and C concern truth according to $\mathcal{N}$, yet the proof of the above in Charlesworth (2006) avoids conflict with Tarski's theorem on the undefinability of truth. That theorem, whose proof is a rigorous reformulation of the liar paradox,

implies that truth (in full generality) according to $\mathcal{N}$ cannot be defined within the language of PA.

The lemma in "Rigorous Analogues for Both Check and CheckMate" then implies that, for each Turing machine P, each of the following five formulas of PA is true:

$$D(P) \rightarrow [\neg C(P, P') \rightarrow HLT(P, P')] \tag{1}$$

$$D(P) \rightarrow B(P, P') \tag{2}$$

$$D(P) \rightarrow [C(P, P') \rightarrow \neg H(P')] \tag{3}$$

$$D(P) \rightarrow [\neg C(P, P') \rightarrow [R(P, P', \neg H(P')) \rightarrow H(P')]] \tag{4}$$

$$D(P) \rightarrow [\neg C(P, P') \rightarrow [R(P, P', H(P')) \rightarrow \neg H(P')]]. \tag{5}$$

For instance, the truth of (3) follows from (b) of the lemma together with the meaning of formula (3), which by expository convention is according to $\mathcal{N}$, and thus is: If a TM decision system P's halting with input $\langle P', Halt?\rangle$ would imply that what P decides about the halting problem for P' is true, then it follows that P' does not halt.

The notation $\mathcal{T}$ denotes the infinite set of true formulas (1) through (5), where P ranges over all Turing machines. Formulas of $\mathcal{T}$ are written using nested implications to support a particular role of *modus ponens* mentioned in "How the Comprehensibility Theorem Fits into the Model", which is the trick that enables the proof of the Comprehensibility Theorem to avoid the kind of dangerously non-rigorous reasoning identified in our section "The Inherent Lack of Rigor in the Preceding "Proof"".

Definition of "Adequate" Formal System

The theorem requires an agent to give its problem-solving proof attempts within an "adequate" formal system $\mathbb{F}$. In order for $\mathbb{F}$ to be **adequate**, the model requires that $\mathcal{T}$ be a subset of the axioms of $\mathbb{F}$, and also that $\mathbb{F}$ satisfy the straightforward requirements in the rest of this paragraph. It must be an extension of PA; that means that each formula (respectively, axiom, rule of inference) of PA is a formula (respectively, axiom, rule of inference) of $\mathbb{F}$. Each sentence of PA that is a formal theorem in $\mathbb{F}$ must be true. Finally, each formula A of $\mathbb{F}$ must have a Gödel code $\#A$ so that the function that maps A to $\#A$ and its inverse are computable, and when A is also in the language of PA, the code for A must be the same as PA's code for A.

Adequate formal systems can differ sharply from one another. The simplest is obtained by adding the formulas of $\mathcal{T}$ to the axioms of PA; its set of theorems is computably enumerable, because the lemma in "Rigorous Analogues for Both Check and CheckMate" guarantees an algorithm that takes any Turing machine P as input and constructs the formulas of $\mathcal{T}$ corresponding to P. Since the formulas of $\mathcal{T}$ are true, our assumption that PA is sound implies the truth of each sentence of PA that is a formal theorem of that system. One example of an adequate system whose

set of theorems is not computably enumerable results from adding all true formulas of PA to the set of axioms of PA. Another example whose set of theorems is not computably enumerable results from extending PA in the natural way by adding the symbol $\in$ and letting the axioms be those of Zermelo-Fraenkel set theory (ZF) together with all true formulas of PA; the requirement about the truth of each sentence of PA that is a formal theorem of this system goes beyond the assumptions needed to prove the Comprehensibility Theorem.

In principle, one can state the Comprehensibility Theorem and carry out its proof within ZF, thereby satisfying the current standard criterion for rigor, without the need for assuming the consistency of ZF or the need for defining within ZF the details of each specific $\mathbb{F}$ that satisfies the requirements for being adequate.

Definition of "Correctly Deduce"

The *nonrigorous* definition of "correctly deduce" in our section "A Conjecture and a Nonrigorous "Proof"" has three parts. The model provides not only a mathematical way to restate those three parts, it supports a fourth part that enables the proof of the theorem to avoid the kind of dangerously non-rigorous reasoning identified in our section "The Inherent Lack of Rigor in the Preceding "Proof"".

Let $\mathcal{A}$ be a function from natural numbers to natural numbers, so that $\mathcal{A}$ models an agent. Let $\mathbb{F}$ be adequate. Let S be a TM decision system; thus S has the property: for each halting problem input of S for which S produces output, that output attempts to settle the halting problem for the corresponding input. The definition of "$\mathcal{A}$ **can correctly deduce correctness-related properties of** S" has four parts, where $\mathcal{A}$ gives its correct proofs within $\mathbb{F}$:

- Part (1). For the halting problem related to any software P: $\mathcal{A}$ can correctly deduce whether or not S makes an incorrect decision and $\mathcal{A}$ can provide a correct proof to support that deduction.
  More precisely, $\mathcal{A}(\langle S, P, Correct?\rangle)$ *yields a formal proof in* $\mathbb{F}$ *either of* $C(S,P)$ *or of* $\neg C(S,P)$. Saying $\mathcal{A}$ with a certain input "yields a formal proof" of some formula A means that $\mathcal{A}$'s output is the code for a list of codes of formulas such that that list is a formal proof whose last formula is A. Here $\langle S, P, Correct?\rangle$ denotes Turing machine input that can be computationally decoded[3] to yield S and P together with an indication that the relevant problem is partial correctness, mentioned in "A Set of Formulas Supporting a Technical Trick".
- Part (2). For the halting problem related to any software P: if S produces output for that problem, then $\mathcal{A}$ can correctly deduce that output and $\mathcal{A}$ can provide a correct proof to support that deduction (if such a correct proof is possible in $\mathbb{F}$).
  More precisely, *if there is a formal proof in* $\mathbb{F}$ *either of* $R(S,P,H(P))$ *or of* $R(S,P,\neg H(P))$, *then* $\mathcal{A}(\langle S, P, Result?\rangle)$ *yields such a formal proof.*

[3] As explained in footnote 1, one can choose to use prime-power coding, so that $\langle S, P, Correct?\rangle$ denotes $2^{\#S}3^{\#P}5^2$. That last exponent is 2 since *Correct*? represents the constant 2. Similarly, one can code the three other halting-related problems, mentioned shortly, by letting *Result*? represent 3, *Decision*? represent 4, and *BreakTie*? represent 5.

- Part (3). $\mathcal{A}$ can correctly deduce (and correctly prove its deduction) that for each halting problem input of S for which S produces output, that output attempts to settle the halting problem for the corresponding input.
  More precisely, $\mathcal{A}(\langle S, Decision? \rangle)$ *yields a formal proof in* $\mathbb{F}$ of $D(S)$. (Recall that $D(S)$ is indeed true, since S is a TM decision system.)
- Part (4). For each P, the agent $\mathcal{A}$ can also obtain additional correct deductions about S and P requiring no more than three additional applications of *modus ponens*.
  More precisely, *for each* P, *consider the set containing the axioms in* $\mathcal{T}$ *and the (at most three) formal theorems related to* S and P *resulting from Parts (1) through (3). Whenever it is possible to obtain a formal theorem about* S *and/or* P *from that set using at most three additional applications of modus ponens*, $\mathcal{A}$ *can yield a proof of that formal theorem*. By a formal theorem "about S and/or P" is meant a formal theorem $H(P)$ or $\neg H(P)$ or a formal theorem that results from substituting S and P, in that order, into B, $\neg B$, HLT, or $\neg HLT$.

Part (4) requires, for each P, that $\mathcal{A}$ can generate a few algorithmically-obtainable (see "Rigorous Analogues for Both Check and CheckMate") members of $\mathcal{T}$ and that $\mathcal{A}$ can at least obtain results related to S that a system using relatively modest mechanical deductions could deduce.

It is the role that $\mathcal{T}$ plays in Part (4) that permits the proof of the Comprehensibility Theorem to avoid the fallacious kind of self-reference identified in our section "The Inherent Lack of Rigor in the Preceding "Proof"". Also, when an agent $\mathcal{A}$ satisfying the above definition makes a deduction, the axioms in $\mathcal{T}$ are there to help. Thus $\mathcal{A}$ need not derive—from the axioms of PA—the formulas of $\mathbb{F}$ contained in $\mathcal{T}$, which are true according to $\mathcal{N}$ regardless of the specific S. This is analogous to the fact that $\mathcal{A}$ need not derive—from more primitive axioms than those of PA—the first-order induction formulas, which are axioms of PA and which are true according to $\mathcal{N}$. The fact that the members of $\mathcal{T}$ fail to be axioms of PA merely reflects the needs of the usual applications of PA.

How the Comprehensibility Theorem Fits into the Model

Here is the mathematical statement of the theorem.

> Comprehensibility Theorem: *Let* $\mathcal{A}$ *any agent*, S *any TM decision system, and* $\mathbb{F}$ *any adequate formal system within which all formal proofs are to be given.*
>
> *If* $\mathcal{A}$ *can correctly deduce correctness-related properties of* S, *then* $\mathcal{A}$ *can correctly deduce* S *does not have* $\mathcal{A}$*'s capability for solving halting problems.*

The theorem's statement uses the preceding mathematical definitions plus the phrase "$\mathcal{A}$ can correctly deduce that S does not have $\mathcal{A}$'s capability for solving halting problems". That phrase is roughly defined to mean there is an instance of the CHESS game for which $\mathcal{A}$ can make (and prove the correctness of) a correct

decision and also can give a correct proof that S cannot make a correct decision. More precisely, that phrase is defined mathematically to mean *there exists a Turing machine P such that $\mathcal{A}(\langle P, Halt?\rangle)$ yields a formal proof of the true one of $H(P)$ and $\neg H(P)$, and in addition, such that $\mathcal{A}(\langle S, P, BreakTie?\rangle)$ yields a formal proof of $B(S, P)$.*

We can now sketch the theorem's proof that appears in Charlesworth (2006) as follows: Let $\mathcal{A}$ and S satisfy its hypothesis, and take P in the definition of "can correctly deduce correctness-related properties" to be S'. Applying the relationship between $\mathcal{T}$ and Part (1) through Part (3), one can use Part (4) to prove that three applications of *modus ponens* enable $\mathcal{A}$ to generate the formal proofs needed to satisfy the theorem's conclusion.

The Comprehensibility Theorem is an implication relating two kinds of deducing: roughly, if an agent $\mathcal{A}$ deduces some correctness-related properties of S, then $\mathcal{A}$ also deduces an intelligence-related limitation of S. The theorem requires correctness of $\mathcal{A}$ as that relates just to the specific S. All other formal proof attempts by $\mathcal{A}$ can be incorrect, and there can be infinitely many such attempts. But $\mathcal{A}$'s deductions about S must be correct regardless of how many inputs to S affect the output of S, and for some S there are infinitely many such inputs. [ES10 explains how the correctness requirement on $\mathcal{A}$ could be greatly reduced.]

Applying the Comprehensibility Theorem to Real-World Agents

This section examines the application of the Comprehensibility Theorem to real-world agents, *without requiring that such agents satisfy computability or infallibility properties*. The description in this section is unconventionally precise. Our later section "Is it Necessary to Expect an Agent to Give Formal Proofs?" employs the kind of less-precise terminology commonly used in real-world situations.

Turing's Assumption About Real-World Agents

Like any other real-world application of mathematics, such as applying a theorem about differential equations to the flight of a rocket, the application strategy we describe cannot be fully mathematical, since no one has ever given a fully mathematical definition of any real-world object. Similarly, in explaining how the kind of abstract mathematical machine Alan Turing defined captures essential aspects of the real-world concept of computation, he made the following non-mathematical assumption, where by "computing" he meant computing by real-world (in particular, human) agents:

> Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-

> dimensional paper, i.e., on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. (Turing 1936)

For convenience, we adopt Turing's assumption that there exists such a one-dimensional paper tape in the real-world. Without loss of generality we further assume there are just two kinds of symbols that may be printed on the paper. To the extent that space on the paper tape permits, the unary code for a natural number can be printed on the paper, with multiple such codes separated by single blank squares. Also, without loss of generality, we may further assume that one of the squares of the paper is designated as the origin, so that (to the extent that space on the paper tape permits) the list of input numbers appears in order left-to-right as a FIFO queue to the right of that square, and the list of output numbers appears in order right-to-left as a FIFO queue to the left of that square.

Notice that here we are not using such conventions as they are routinely used when employing abstract mathematical Turing machines to prove theorems in the theory of computation. Instead, here we are pointing out that a particular generally-accepted assumption of Turing (which we adopt for convenience) justifies the existence *in the real-world* of a physical system that receives a given natural number from a FIFO input queue and permits the recording of a corresponding official output so there is a FIFO output queue.

In our subsequent description of the use of the two queues, we do not assume that the one-dimensional real-world paper tape described by Turing is infinite in extent. There will be no problem with our subsequent description, for instance, if there are single numbers too large to appear as inputs, and there will be no problem if there is inadequate space for some or all of the output.

Applying Turing's Assumption to the Comprehensibility Theorem

Let A denote a single agent, in the real-world sense of "agent"; that is, let A be a real-world system that consists of one or more robots, supercomputers, humans, humans with surgically-implanted silicon memory chips, and so forth.

Here is how A is modeled as a function $\mathcal{A}_A$. The set A receives from the input queue a natural number n, where n is a code for a halting problem. For simplicity, this paragraph assumes this is the first input equal to n received by A. Until A subsequently produces two official outputs in the required form that correspond to n (additional details are given later in this paragraph), the set A is not permitted to receive any additional input. After receiving n, the set A can attempt to decode n to obtain the statement of the problem to solve; it is not necessary to assume A accomplishes such decodings flawlessly. When—if ever – A is ready to indicate a result, it produces two official outputs: the value of n immediately followed by the value of A's attempted code for an attempted formal proof to settle the particular problem.

The value of $\mathcal{A}_A(n)$—if it exists—is the official natural number output—if there is such an output—produced by A that immediately follows the *first* output of n produced by A, after the *first* input equal to n received by A. Because of the requirements related to "*first*", even if A "changes its mind" (whatever that might

mean) and produces—after its first input equal to n—numerous official outputs paired with n at different times, $\mathcal{A}_A$ as just defined is a mathematical function. This application strategy defines a function $\mathcal{A}_A$ from a (possibly empty) subset of the natural numbers to the natural numbers without requiring the computability or infallibility of A, and one can apply the theorem to $\mathcal{A}_A$. (For other possible application strategies, see ES1.)

The description in this section places a priority on clarity, through the use of a highly restrictive rule that prohibits A from receiving any input during the time interval between when it first receives the input n and when it subsequently produces its two official outputs corresponding to n. This rule prevents A from "solving" the problem related to n by simply using the content of an additional input during the time it is trying to solve the given problem. Of course, such an act would not be compatible with the intuitive meaning of "correctly deduce". To notice the kind of difficulty avoided here, observe that the natural number code for a formal proof that solves the problem related to n could appear as a number used within a particular Turing machine related to another problem that is coded in an additional natural number input.

A Closer Look at the Preceding Description

We now examine more closely what it means for some of the assumptions mentioned above to be satisfied in real-world situations. First note that the definition of a mathematical agent used in the theorem is an abstract function, so it is only necessary to show how to obtain a mathematical function—rather than a computable function—from a real-world situation.

Note that there is no effect on the value $\mathcal{A}_A(n)$ if, between receiving its first input of n and producing the two official corresponding outputs in the required form, the set A produces other official outputs related to other work accomplished by A, as long as no such additional output affects directly or indirectly—such as by causing the interleaving of digits of two official outputs—either the first official output of A equal to n itself or the official output that immediately follows the first official output of A equal to n.

Also note that the Comprehensibility Theorem, like any theorem, can be written in the form (p implies q). Each application of such a theorem can be written as:

h implies ($\bar{p}$ implies $\bar{q}$),
and as:
(h and $\bar{p}$) implies $\bar{q}$,

where hypothesis h is the conjunction of some necessarily non-mathematical assumptions about the real-world situation, and where $\bar{p}$ and $\bar{q}$ are the interpretations of p and q in the real-world situation. To apply the Comprehensibility Theorem, one states within h the real-world assumptions specific to the constituents of A and the functioning of the input and output queues. One obtains $\bar{p}$ and $\bar{q}$ by replacing agent $\mathcal{A}$ in each of p and q by $\mathcal{A}_A$. The effect is to rephrase statements about values of $\mathcal{A}$ as statements about A and the input and output queues.

Next note that in any situation for which (h and $\bar{p}$) is true, the conclusion $\bar{q}$ is also true, regardless of whether any entity is aware that (h and $\bar{p}$) is in fact true. The nebulous concept of "awareness" is factored out of the Comprehensibility Theorem. This is, of course, the case for customary applications of mathematics. One can apply a theorem about differential equations to the astronomy of an unobservable region of the universe: the (h and $\bar{p}$) in such an application can be true, and consequently the $\bar{q}$ can be true, regardless of whether any entity is aware of the truth value of (h and $\bar{p}$).

We now examine more closely the restrictive rule that prohibits A from receiving any input during the time interval between when it first receives the input n and when it subsequently produces its two official outputs corresponding to n. Although some real-world situations might permit this rule to be enforced in a computable way, that is not actually required. Outputs that violate the restrictive rule simply fail to be official outputs, and this is the case regardless of whether any entity is aware whether or not any particular output is official. As an example, here is a paraphrase of the application of Part (1) of the definition of "correctly deduce" to a specific A, S, and P: if, upon receiving its first input of $\langle S, P, Correct? \rangle$ and before receiving any further inputs the real-world agent A produces the output $\langle S, P, Correct? \rangle$ and its first such output is immediately followed by an output that is the code for the particular kind of list of codes specified in Part (1), then in performing that action $A(\langle S, P, Correct? \rangle)$ "yields a formal proof in $\mathbb{F}$ either of $C(S, P)$ or of $\neg C(S, P)$".

One can apply the theorem to A, regardless of the real-world systems constituting A. If A fails to satisfy the $\bar{p}$ for a theorem, that theorem holds by default for A. (Some practical agents that satisfy the hypothesis of the theorem are considered in ES3.) But one must interpret the theorem with care. For example, let $H \cup C$ consist of a specific set of humans and a specific computer, where C could assist in the—not necessarily infallible—decoding and encoding, it could facilitate developing formal proofs from nonformal proofs, and it could check cases that arise within a proof, like the extensive computerized case-checking in the first proof of the Four Color Theorem (Appel and Haken 1976), which solves a halting problem as pointed out in "Some Results and Conjectures That Are Halting Problems". If $\mathcal{A}_{H \cup C}$ is used in the hypothesis of the theorem, it (and not $\mathcal{A}_H$) must be used in the conclusion of that application as well. The Comprehensibility Theorem gives: "if $\mathcal{A}_{H \cup C}$ can correctly deduce correctness-related properties of S, then $\mathcal{A}_{H \cup C}$ can correctly deduce that S does not have $\mathcal{A}_{H \cup C}$'s capability for solving halting problems".

Questions About the Model and the Comprehensibility Theorem

Since they are quite different from standard uses of formal logic, the model and theorem raise many questions. The preceding part of this article is essential to facilitate this section and the Electronic Supplement, which address over twenty such questions. To the best of the author's knowledge, all sections address previously-unaddressed questions, except for sections "How Important is Eliminating Infallibility Assumptions About Intelligent Agents?", "Where Does the Mathematical Model Avoid Infallibility Assumptions About Agents?", "How Can

One Assume that a System Used by an Agent is Sound Without Assuming Agent Infallibility?", "Is An Agent that Makes Mistakes Necessarily Trivial?", and ES3 (of the Electronic Supplement), whose questions were addressed in Charlesworth (2006). Some new explanations appear in each section. *Occasionally a discussion here briefly repeats something elsewhere in this section, to facilitate skipping to questions of most interest to the individual reader.*

How Important is Eliminating Infallibility Assumptions About Intelligent Agents?

The use of algorithms known to be fallible pervades Artificial Intelligence, often because of the intractibility of many problems that arises from either the unsolvability of the halting problem or NP-completeness. One measure of the pervasiveness of such algorithms is that the index of Russell & Norvig's AI textbook lists a dozen places in the book where the word "heuristic" plays an important (i.e., boldface) role. Also, the Philosophical Foundations chapter of that book states:

> ... if anything, humans are known to be inconsistent. This is certainly true for everyday reasoning, but it is also true for careful mathematical thought. A famous example is the four-color map problem. Alfred Kempe published a proof in 1879 that was widely accepted and contributed to his election as a Fellow of the Royal Society. In 1890, however, Percy Heawood pointed out a flaw and the theorem remained unproved until 1977 (Russell and Norvig (2010), p. 1023).

Alan Turing went so far as to assert "if a machine is expected to be infallible, it cannot also be intelligent" (Turing 1986 p. 124, quote dated 1947). More recently, Marvin Minsky (Minsky (1995)) has asserted "There's no reason to assume ... that either human minds or computing machines need to be perfectly and flawlessly logical".

Regardless of whether one thinks a highly intelligent, infallible agent can or cannot exist, it is advantageous—when possible—for a theorem to be independent of that issue, since such a theorem is a more general result. The Comprehensibility Theorem achieves that kind of independence.

Before leaving the above question, we should point out that the assertions of Turing and Minsky above were made in the context of their strong criticism of some arguments whose goal was to show that software-based human-level AI is impossible. That goal is not the purpose, and is not achieved by, the Comprehensibility Theorem.

Where Does the Mathematical Model Avoid Infallibility Assumptions About Agents?

[This part is indebted to Charlesworth (2006) to a much greater extent than the other parts of section "Questions About the Model and the Comprehensibility Theorem" and the Electronic Supplement.] The model defines an agent's "decision" about a

halting problem without requiring the correctness of that decision; that is, without requiring the truth (according to $\mathcal{N}$) of the formula that indicates that decision. The model does not require that a proof attempt by an agent must be logically correct. The model does not assume that an agent infallibly knows when it has given a correct decision or made a logically correct proof attempt. The model's definition of "correctly deduce" and the proof of the Comprehensibility Theorem are based on the lemma in "Rigorous Analogues for Both Check and CheckMate", which does not assume that even 1 % of a Turing machine's decisions about halting questions are correct. The model's definition of "S has $\mathcal{A}$'s capability for solving halting problems" could be satisfied even if S makes incorrect decisions about infinitely many halting problems. The fact that a real-world agent's only response that counts is its *first* official response (see section "Applying the Comprehensibility Theorem to Real-World Agents" and ES1) permits a real-world agent to make blatantly contradictory decisions about the halting of a specific Turing machine, as is also the case for other questions given to the agent. Our section "Applying the Comprehensibility Theorem to Real-World Agents" also explains that the model does not assume an agent decodes an input number flawlessly; there is an algorithm for the decoding, but the model does not assume the agent carries out that algorithm flawlessly.

The lack of infallibility assumptions on agents is discussed near the end of ES3 in the context of sketching a particular real-world application of the theorem.

How Can One Assume that a System Used by an Agent is Sound Without Assuming Agent Infallibility?

The importance of avoiding infallibility assumptions is reviewed in our section "How Important is Eliminating Infallibility Assumptions About Intelligent Agents?".

The model assumes that Peano arithmetic, PA, is sound under the standard interpretation $\mathcal{N}$ of the symbols of PA, an assumption widely-accepted among mathematicians and computer scientists. (The relevant definition of sound is in our section "Gödel's Second Incompleteness Theorem is a Halting Problem Result".) Moreover, the Comprehensibility Theorem requires an agent to provide its deductions using an "adequate" formal system $\mathbb{F}$, and one required property of such a system is that each sentence of PA that is a formal theorem in $\mathbb{F}$ must be true under the interpretation $\mathcal{N}$.

This requirement on $\mathbb{F}$ does not imply the infallibility of an agent. That is because the model distinguishes between $\mathbb{F}$ and the agent. Even if $\mathbb{F}$ has perfect properties, an agent $\mathcal{A}$ that uses $\mathbb{F}$ can fall far short of being infallible. (Two examples of fallible agents are in our section "Rigorous Analogues for Both Check and CheckMate" and an additional example is discussed near the end of ES3.) That is analogous to saying the rules of the usual board game of chess have infallibility properties—such as implying that after each legal move no square contains two pieces—yet also saying a player of chess can be fallible and try to violate those rules, with such an attempt simply being called "illegal" (i.e., incorrect).

How Does the Comprehensibility Theorem Differ from Gödel's Theorem and the Unsolvability of the Halting Problem?

Gödel's Incompleteness Theorem is about the incompleteness of any sufficiently-powerful, consistent, formal axiomatic system whose axioms are computably enumerable and whose rules of inference are computably applied. Turing's unsolvability of the halting problem is about a limitation that any Turing machine has for solving all instances of the halting problem infallibly.

Rather than being about incompleteness or unsolvability, the Comprehensibility Theorem is about an agent's ability to make deductions about software. It identifies a relationship between an agent's ability to deduce a correctness property of software and that agent's ability to deduce an intelligence-related limitation of that same software.

Also, the results of Gödel and Turing apply only to certain kinds of *infallible* systems, whereas the Comprehensibility Theorem can be applied to systems (i.e., to agents) that are far from infallible. (See "Where Does the Mathematical Model Avoid Infallibility Assumptions About Agents?".) The halting problem limitation is a limitation on a Turing machine that is assumed to be infallible. Concerning Gödel's Incompleteness Theorem, there is typically extreme brittleness associated with the concept of the consistency of a formal system. A standard formal system extending PA is consistent only if *there is not even a single* sentence A in the system such that both A and its negation $\neg A$ are formal theorems. Moreover, such a system is trivial in a typical formal system if that system is inconsistent, since in a typical system if A and $\neg A$ are formal theorems, then $A \vee B$ is a formal theorem where B is *any sentence* (regardless of whether the sentence is false) of PA and $\vee$ is the logical "or" operator, and it then follows from the theoremhood of $\neg A$ that B is a formal theorem.

Also, the Comprehensibility Theorem can be applied to much broader kinds of systems than can Gödel's and Turing's results. An agent need only be a function from natural numbers to natural numbers, rather than having to satisfy more-restrictive properties. (See "Applying the Comprehensibility Theorem to Real-World Agents".) There are uncountably-many functions from natural numbers to natural numbers, far more than the countably-many such functions defined by Turing machines. Likewise, Gödel's Theorem is about formal axiomatic systems whose axioms and rules of inference are computable. The Comprehensibility Theorem can be applied to human agents, without assuming anything about either the computability or the infallibility of humans.

One important property shared by the Comprehensibility Theorem and those results of Gödel and Turing is that each is a mathematical theorem satisfying the current criterion for rigor: each is provable in principle in Zermelo-Frankel Set Theory, rather than merely being the result of a non-rigorous argument. That is unusually important because the proof of each uses some form of self-reference. Non-rigorous reasoning using self-reference can easily produce self-contradictory results. (See "The Inherent Lack of Rigor in the Preceding "Proof"".)

Is An Agent that Makes Mistakes Necessarily Trivial?

This question arises because any standard formal system extending PA that is inconsistent is trivial, since such a system permits the deduction of each sentence of PA, regardless of whether the sentence is true or false (see "How Does the Comprehensibility Theorem Differ from Gödel's Theorem and the Unsolvability of the Halting Problem?"). But the way the mathematical model underlying the theorem distinguishes between the agent $\mathcal{A}$ and the formal system $\mathbb{F}$ implies that the answer to the above question is "no". *Any* natural-number valued function defined on a subset of the natural numbers is an agent; an agent need have no other property. In particular, in order for "$\mathcal{A}$ can correctly deduce correctness-related properties of S" to hold, $\mathcal{A}$ need only satisfy the requirements specifically related to S in the definition of that concept; there are no restrictions on the values (if any) of the function $\mathcal{A}$ for other inputs. Thus, if such an $\mathcal{A}$ makes an incorrect decision (that counts in the sense of section "Where Does the Mathematical Model Avoid Infallibility Assumptions About Agents?")—including making opposite decisions about two Turing machines that differ in an utterly insignificant way so that $\mathcal{A}$'s decisions would be inconsistent in the intuitive sense—that does not imply that any of $\mathcal{A}$'s additional decisions about *other* Turing machines must be incorrect.

Are Deductions About Scientific Phenomena Related to Deductions About Computer Programs?

The definitions of "correctly deduce" used in the Comprehensibility Theorem relate to the ability an agent might have of deducing properties of a computer program. The notion of scientific deductions seems much broader than deducing properties of computer programs.

However, scientific deductions related to a natural phenomenon typically involve a reductionist process that attempts to analyze the phenomenon into primitive basic principles, explaining the phenomenon via a systematic, algorithmic procedure based on the resulting primitive principles. One can explain that reductionist process as an application of a viewpoint expressed by Donald Knuth—when that viewpoint is taken literally. Knuth asserts that we don't fully understand something until we teach it to a computer, by writing an algorithm to do it. In Knuth's words:

> A person well-trained in computer science knows how to deal with algorithms: how to construct them, manipulate them, understand them, analyze them. This knowledge is preparation for much more than writing good computer programs; it is a general-purpose mental tool that will be a definite aid to the understanding of other subjects, whether they be chemistry, linguistics, or music, etc. The reason for this may be understood in the following way: It has often been said that a person does not really understand something until after teaching it to someone else. Actually a person does not *really* understand something until after teaching it to a *computer*, i.e. expressing it as an algorithm ... An attempt to formalize things as algorithms leads to a much

> deeper understanding than if we simply try to comprehend things in the traditional way. (Knuth 1996 pp. 9–10), based largely on Knuth (1973).

In the passage indicated by the ellipsis (...) in that quotation, Knuth cites the following comment of George Forsythe: "The automatic computer really *forces* that precision of thinking which is alleged to be a product of any study of mathematics" Forsythe (1959), emphasis in original.

We would add that the computer forces such precise and thorough thinking, not just because of the rigorous language used in computer programs, but because of a bare computer's lack of background, common sense knowledge. We refer to the following assertion as **Knuth's Thesis**: *Fully understanding a phenomenon in the physical universe requires, among other things, being able to make deductions about a computer program that exists and accurately simulates that phenomenon*. If one is willing to assume Knuth's Thesis, the answer to the question in this section's title is "yes".

Knuth's Thesis might be similar to an assertion by Richard Feynman that has been interpreted in terms of derivations; i.e., in terms of deductions. According to Lynch and Granger (2008): "Computational science has as its goal to understand phenomena sufficiently well to reconstruct them ... the same principle was adhered to by the well-known physicist Richard Feynman, on his blackboard at the time of his death in 1988 was this statement: '*What I cannot create, I do not understand*'." [Emphasis added.] Quora (www.quora.com) asked readers to resolve the apparent contradiction between that Feynman assertion and the fact that he was a theoretical physicist. More than two years later the highest-ranked explanation was: "When Feynman said 'create', he ... meant that, starting with a blank piece of paper and the knowledge already in his mind, he could" take a "theoretical result and re-derive it". (Written by Mark Eichenlaub, December 4, 2011; no response disagreed with that response, which received 637 of the 701 signed-by-name, net up-votes for all contributed responses by June 16, 2014.)

This article does not take a position for or against Knuth's Thesis. When we consider what Knuth's Thesis implies, that hypothesis is made explicit.

Is Gödel's Second Incompleteness Theorem Mathematically Relevant to the Mind?

In addition to requiring the strong infallibility assumption of consistency, Gödel's Second Incompleteness Theorem—which is stronger than his Incompleteness Theorem—is not *mathematically* relevant to the mind, according to specialists on that theorem. In *Gödel, Escher, Bach*, Douglas Hofstadter points out that the assertion "we cannot understand our own minds/brains"—which is a central theme of Hofstadter's book—*cannot be obtained as an application of that theorem* and is merely a "metaphorical analogue to Gödel's Theorem" Hofstadter (1979), p. 697. The accuracy of Hofstadter's observation is praised in a book pointing out many misuses of Gödel's Theorem, whose author Torkel Franzén applauds this specific passage of Hofstadter for "the virtue of making it explicit that the role of the

incompleteness theorem [to obtain an assertion about the human mind/brain] is a matter of inspiration rather than implication" (Franzén 2005 p. 124).

Is There Only a Restriction on an Agent's Deductive Self-comprehensibility?

The Comprehensibility Theorem can be paraphrased as saying that if an agent can correctly deduce correctness-related properties of software *S*, then that agent can correctly deduce that *S* does not have *that agent's* capability for solving halting problems.

The Comprehensibility Theorem appears to be the first mathematical theorem (see "Is Gödel's Second Incompleteness Theorem Mathematically Relevant to the Mind?") implying the impossibility of any AI agent or natural agent (including a not-necessarily infallible and not-necessarily computable human agent) satisfying a rigorous and deductive interpretation of the Delphic challenge "know thyself", if one assumes Knuth's Thesis concerning what is necessary for fully understanding any natural phenomenon (see the section "Are Deductions About Scientific Phenomena Related to Deductions About Computer Programs?"). Even according to presumably a less rigorous interpretation of that self-comprehensibility challenge, Socrates—according to Plato—admitted not being able to satisfy the challenge, yet considered that challenge to be among the most important of intellectual tasks; see Plato's Phaedrus 230 (Hamilton and Cairns 1961 p. 478). Likewise, America's first widely-known scientist—Benjamin Franklin—considered that challenge "extremely hard" to satisfy; see passage 521 of Franklin (1914).

The answer to the above question appears to be "yes", if one is limited to the Comprehensibility Theorem. Similarly, if one is limited to the discovery of Gödel's Incompleteness Theorem in 1931, then the only known restriction on a formal logical system would be a restriction on the provability of a formal statement that *merely expresses a property of that logical system itself*. As a result, for many years after 1931 perhaps the vast majority of research mathematicians saw no relevance of the Incompleteness Theorem to the mathematical conjectures they sought to settle via proofs or counterexamples. In the decades since 1931, however, restrictions on the provability of numerous mainstream mathematical—as distinct from logically self-referent—conjectures within PA or ZF or ZFC have been discovered. Here are two examples. Goldstein's Theorem, which is provable in ZFC and relates to expressing natural numbers using exponents and bases for exponents (distinct from the use of such notations within formal systems), is expressible in PA but not provable within PA; Kirby and Paris (1982). The Continuum Hypothesis, which is related to the possible existence of a set whose cardinality is strictly between that of the set of natural numbers and that of the set of real numbers (see "Hilbert's Thesis Implies Each Rigorous Math Conjecture is a Halting Problem"), is expressible in ZF but neither provable nor disprovable in either ZF or ZFC; Gödel (1939), Cohen (1963). The proofs of such results can be altogether different from Gödel's proof of the Incompleteness Theorem. For instance, Cohen's proof used his invention of the entirely new methodology related to ZF known as "forcing".

Historical note: Sect. 3 of Gödel (1931) showed that the kind of unprovable formal statement identified in that article is equivalent to a statement about the

existence of solutions to polynomial equations over the integers; also see (Wang 1987 p. 279). The resulting statement about polynomials (merely) expresses a property of the logical system in question, since it is equivalent to the kind of unprovable formal statement identified in that article, which itself has that property.

If one is willing to assume Knuth's Thesis, one can say the following. The impossibility of deductive self-comprehensibility by an artificial or natural agent is a result in applied mathematics; that is, it is an application of a mathematical theorem, rather than a philosophical assertion. Moreover, it is independent of the controversial issue of whether an accurate computer simulation of that agent is possible: if such a simulation is not possible, then the impossibility of deductive self-comprehensibility follows from Knuth's Thesis directly; otherwise, the impossibility of deductive self-comprehensibility follows from an application of the Comprehensibility Theorem. Furthermore, since the agent mentioned in the Comprehensibility Theorem is not-necessarily infallible, the impossibility of deductive self-comprehensibility by the agent is independent of the controversial issue of whether one can view that agent as infallible.

Before leaving the question addressed in this section, we suggest that self-comprehensibility in some form might be essential for a kind of self-reflection useful for self-improvement that enables some agents, such as some human agents, to increase their success. That may be the reason Socrates ranked self-comprehensibility "among the most important of intellectual tasks".

Is it Necessary to Expect an Agent to Give Formal Proofs?

It was essential to use *formal* proofs as objects within the model, since one must use self-reference within proofs with extreme care as explained in our section "The Inherent Lack of Rigor in the Preceding "Proof"". Now that there is a proof of the Comprehensibility Theorem satisfying the current standard criterion for rigor, one can carefully *apply* it non-formally without risking paradoxical results.

The description of applying the theorem given in our section "Applying the Comprehensibility Theorem to Real-World Agents" attempts to be precise. We now employ less-precise terminology customarily used by computer scientists, placing such terminology inside double quotes. One can apply the theorem to real-world situations in which the "proofs" of programmers or of any agents are not actually presented within a formal axiomatic system *as long as it is possible—in principle—to do so*, and in which the texts of programs are typically not in terms of Turing machines and the texts of programs and "proofs" are typically not represented by numerical codes. Recent practical software systems that discover proofs of termination and non-termination are discussed in ES3; those proofs are not within a formal system, but one can consider them as the kind of "proofs" mentioned in this paragraph. As another kind of example, consider any AI agent that produces non-formal deductions—via a potentially unending search through any search space—where the agent also produces the path to any solution that it finds; one can view such a path as a "proof" of the corresponding deduction. Furthermore, one can relax restrictions of section "Applying the Comprehensibility Theorem to Real-World Agents" to permit a real-world agent to receive any input at any time, as long

as an input does not provide "a hint or complete solution" to a halting problem that agent is currently trying to solve.

Eliminating formal proofs in applications of the Comprehensibility Theorem does not undermine the use of formal proofs in our section "Hilbert's Thesis Implies Each Rigorous Math Conjecture is a Halting Problem" in showing that Hilbert's Thesis implies broad importance to halting problems, since one can translate a non-formal conjecture into a formal conjecture and obtain related software, such that answering the halting problem for that software is equivalent to answering the non-formal conjecture. Also, a recent article suggests the increasing and long-term importance of formal proofs in both mainstream mathematics and formal verification of software Harrison (2008).

Is the Comprehensibility Theorem Relevant to the Foundations of AI?

The opening paragraph of this article asserts that problem-solving software that is not-necessarily infallible is central to AI, an assertion that seems reasonable in view of much of the content of an AI textbook like Russell and Norvig (2010). The opening paragraph continues by asserting that such software whose correctness and incorrectness properties are deducible by agents—both artificial and natural agents, and where the agents themselves are not-necessarily infallible—is an issue at the foundations of AI. That additional assertion also appears to be reasonable, perhaps too obvious an assertion to have been made prior to this article. What, then, is the relevance of the Comprehensibility Theorem to the phrases in those assertions? Based on Hilbert's Thesis (accepted in practice by nearly all mathematicians), section "Hilbert's Thesis Implies Each Rigorous Math Conjecture is a Halting Problem" explains that a model of intelligent "problem-solving" has broad scope throughout AI, if it—like the model underlying the theorem—is focused on solving halting problems; this broad nature of halting problems is apparently overlooked in previous AI journal articles. Sections "Definition of "Correctly Deduce"", "Where Does the Mathematical Model Avoid Infallibility Assumptions About Agents?", "How Can One Assume that a System Used by an Agent is Sound Without Assuming Agent Infallibility?", and "Is An Agent that Makes Mistakes Necessarily Trivial?" explain that both the software and the agents mentioned in the theorem are "not-necessarily infallible". Sections "A Set of Formulas Supporting A Technical Trick" and "Definition of "Correctly Deduce"" explain how the statement of the theorem is relevant to "correctness and incorrectness properties" of software. The sections "Applying the Comprehensibility Theorem to Real-World Agents" and "Is it Necessary to Expect an Agent to Give Formal Proofs?" explain how one can apply the theorem to agents of any kind, hence to "both artificial and natural agents". Also, ES3 and ES4 discuss the long-term applicability of the Comprehensibility Theorem to practical AI systems. A summary of ways the theorem is applicable to the foundations of AI will be given in our "Conclusions".

The preceding paragraph presents reasons for a "yes" answer to the question addressed in this section. The Comprehensibility Theorem's impact on the foundations of AI could become more evident in the long-term, assuming mathematicians and software engineers increasingly use formal theorem-provers

as predicted in Harrison (2008), p.1399. But that assumption is not essential here, since section "Is it Necessary to Expect an Agent to Give Formal Proofs?" explains that in *applying* (as distinct from proving) the Comprehensibility Theorem it is not essential to require an agent to give formal proofs.

What Additional Questions are Addressed in the Electronic Supplement?

Those questions are:

ES1: Must real-world applications emphasize the first official response?
ES2: Is truth of the members of $\mathcal{T}$ an infallibility assumption about agents?
ES3: Are there examples for which "correctly deduce" is satisfied?
ES4: What is the relevance of system software to the foundations of AI?
ES5: What about the interchangeability of Turing machines and axiomatic systems?
ES6: Should the mathematical model be so focused on software?
ES7: How relevant are reflection principles?
ES8: How is an agent supposed to have access to the members of $\mathcal{T}$ and the theorems of $\mathbb{F}$?
ES9: What about the difference between the formalization of provability of a mathematical conjecture and the formalization of the conjecture?
ES10: Can the correctness requirements on agents be reduced?
ES11: Is the definition of "correctly deduce" unrealistically strong?

Conclusions

The Comprehensibility Theorem is based on natural-number valued functions, Turing machines, formal systems, and halting problems, which provide a convenient level of abstraction. Yet one can see a connection between the model and practical, not-necessarily infallible, problem-solving systems, as discussed in "Where Does the Mathematical Model Avoid Infallibility Assumptions About Agents?" and "Is it Necessary to Expect an Agent to Give Formal Proofs?", and in (Electronic Supplement) ES3 and ES4. Like the definition of the limit concept and the definite integral concept in Calculus, the mathematical definitions used to support the Comprehensibility Theorem are technical and somewhat complicated, yet arise naturally from a simple context; see "A Game Based on Solving Halting Problems".

The Introduction mentions three long-standing shortcomings of earlier treatments: their assumptions about the infallibility of agents, use of inherently fallacious reasoning, and non-applicability to artificial agents. This article supports the conclusion that the Comprehensibility Theorem overcomes those shortcomings; see "Where Does the Mathematical Model Avoid Infallibility Assumptions About Agents?", The Mathematical Model, and "Applying the Comprehensibility Theorem to Real-World Agents", respectively.

The article also considers the following ways in which the Comprehensibility Theorem is applicable to the foundations of AI:

- Based on Hilbert's Thesis (accepted in practice by nearly all mathematicians), a model of intelligent "problem-solving" has broad scope throughout AI, if it—like the model supporting the Comprehensibility Theorem—is focused on solving halting problems; see "Hilbert's Thesis Implies Each Rigorous Math Conjecture is a Halting Problem".
- One can apply the "if ... then" of the theorem to any agent and to any piece of software that makes decisions about halting problems. The agent can be computational (symbolic, connectionist, Bayesian, or based on evolutionary computing or any other computational techniques) or non-computational. The agent can be artificial or natural or a combination of both. The software can be programmed by humans or generated automatically. There are reasonable reasons to view the hypothesis of the theorem as being satisfied by an example in ES3 based on recent practical research by others.
- As explained in our section "Is it Necessary to Expect an Agent to Give Formal Proofs?", since the theorem has a mathematical proof (that uses formal proofs as objects to avoid well-known paradoxical reasoning), one can apply the theorem in real-world applications without requiring the applications to involve formal proofs.

The just-given summary suggests ways in which the Comprehensibility Theorem is a general result. However the Comprehensibility Theorem also appears to be limited in scope, being only a restriction on an agent's deductive self-comprehensibility, as pointed out in our section "Is There Only a Restriction on an Agent's Deductive Self-comprehensibility?". That section mentions that Gödel's Theorem in 1931 appeared to have a similar kind of limited scope, but the kind of incompleteness result revealed by Gödel's Theorem stimulated the subsequent discovery of restrictions on the provability of numerous mainstream mathematical conjectures. Our section "Is There Only a Restriction on an Agent's Deductive Self-comprehensibility?" also suggests that self-comprehensibility in some form might be essential for a kind of self-reflection useful for self-improvement that enables some agents, such as some human agents, to increase their success at problem solving.

In addition, our section "Is There Only a Restriction on an Agent's Deductive Self-comprehensibility?" observes that, if one is willing to assume a thesis due to Donald Knuth, the Comprehensibility Theorem appears to be the first mathematical theorem implying the impossibility of any AI agent or natural agent—including a not-necessarily infallible human agent—satisfying a rigorous and deductive interpretation of the self-comprehensibility challenge. Some have pointed out the difficulty of self-comprehensibility, even according to presumably a less rigorous interpretation. This includes Socrates, who considered that challenge to be among the most important of intellectual tasks. Moreover, the impossibility just mentioned

is a result in applied mathematics, in the sense that it is an application of a mathematical theorem.

Acknowledgments The author thanks Josephine A. Owenby, Patrick E. Burns, Gregory F. Sam, and Mark R. Waser for encouragement, and Betty B. Tobias for library-related support. The author is very grateful for referee suggestions that improved the clarity of this article.

References

Appel, A. W. (1998). *Modern compiler implementation in Java*. New York: Cambridge University Press.

Appel, K., & Haken, W. (1976). Every planar map is four colorable. *Bulletin of the American Mathematical Society*, *82*, 711–712.

Burgess, J. P. (1992). Proofs about proofs: A defense of classical logic. Part I: The aims of classical logic. In *Proof, logic and formalization* (pp. 8–23). London: Routledge.

Charlesworth, A. (2006). Comprehending software correctness implies comprehending an intelligence-related limitation. *ACM Transactions on Computational Logic*, *7*, 590–612.

Cohen, P. (1963). The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences of the USA*, *50*, 1143–1148.

Cook, B., Podelski, A., Rybalchenko, A. (2006). Termination proofs for systems code. In *Proceedings of the 2006 ACM SIGPLAN conference on programming language design and implementation*, pp. 415–426.

Dorin, P. M., & Toal, R. J. (1995). Understanding noncomputability. *Communications of the ACM*, *38*(11), 12.

Forsythe, G. E. (1959). The role of numerical analysis in an undergraduate program. *American Mathematical Monthly*, *66*(8), 651–662.

Franklin, B. (1914). *Poor Richard's Almanac*. Waterloo, IA: The U.S.C. Publishing Co.

Franzén, T. (2005). *Gödel's Theorem: An incomplete guide to its use and abuse*. Wellesley, MA: A. K. Peters.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, *38*, 173–198.

Gödel, K. (1939). Consistency proof for the generalized continuum hypothesis. *Proceedings of the National Academy of Sciences of the USA*, *25*, 28–32.

Gupta, A., Henzinger, T. A., Majumdar, R., Rybalchenko, A., & Xu, R.-G. (2008). Proving non-termination. In *Proceedings of the 2008 ACM SIGPLAN-SIGACT principles of programming languages conference*, pp. 147–158.

Hamilton, E., & Cairns, H. (1961). *The collected dialogues of Plato*. Princeton, NJ: Princeton University Press.

Harrison, J. (2008). Formal proof - theory and practice. *Notices of the American Mathematical Society*, *55*(11), 1395–1406.

Hodel, R. E. (1995). *An introduction to mathematical logic*. Boston: PWS Publishing Company.

Hofstadter, D. (1979). *Gödel, Escher, Bach: An eternal golden braid*. New York: Basic Books.

Hofstadter, D. (1982). Can inspiration be mechanized? *Scientific American*, *247*, 18–34.

Kirby, L., & Paris, J. (1982). Accessible independence results for Peano Arithmetic. *Bulletin of the London Mathematical Society*, *14*(4), 285–293.

Knuth, D. E. (1973). Computer science and its relation to mathematics. *American Scientist*, *61*(6), 707–713.

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

LaForte, G., Hayes, P. J., & Ford, K. M. (1998). Why Gödel's Theorem cannot refute computationalism. *Artificial Intelligence*, *104*, 265–286.

Lynch, G., & Granger, R. (2008). *Big brain: The origins and future of human intelligence*. New York: Macmillan.

Mac Lane, S. (1986.). *Mathematics: Form and function*. New York: Springer.

Minsky, M. (1995). Reply to Penrose's "Consciousness involves noncomputable ingredients". In J. Brockman (Ed.), *The third culture* (pp. 256–257). New York: Simon and Schuster.

Moravec, H. (1992). Minds with mobility [an interview]. *Discover*, *13*(11), 104–106.

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Turing, A. M. (1936). On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 42*, Ser. 2, 230–265 (a correction 43 (1937), 544–546).

Turing, A. M. (1986). Lecture to the London Mathematical Society on 20 February 1947. In *A. M. Turing's ACE report of 1946 and other papers* (pp. 106–124). Cambridge, MA: MIT Press.

van Lint, J. H., & Wilson, R. M. (1992). *A course in combinatorics*. New York: Cambridge University Press.

Wang, H. (1987). *Reflections on Kurt Gödel*. Cambridge, MA: MIT Press.

Wiles, A. J. (1995). Modular elliptic curves and Fermat's last theorem. *Annals of Mathematics*, *131*, 443–551.