Minds & Machines (2008) 18:1–15
DOI 10.1007/s11023-007-9082-2

Consistency, Turing Computability and Gödel's First Incompleteness Theorem

Robert F. Hadley

Received: 8 September 2006 / Accepted: 7 November 2007 / Published online: 6 December 2007

Abstract It is well understood and appreciated that Gödel's Incompleteness Theorems apply to sufficiently strong, formal deductive systems. In particular, the theorems apply to systems which are adequate for conventional number theory. Less well known is that there exist algorithms which can be applied to such a system to generate a *gödel-sentence* for that system. Although the generation of a sentence is not equivalent to proving its truth, the present paper argues that the existence of these algorithms, when conjoined with Gödel's results and accepted theorems of recursion theory, does provide the basis for an *apparent paradox*. The difficulty arises when such an algorithm is embedded within a computer program of sufficient arithmetic power. The required computer program (an AI system) is described herein, and the paradox is derived. A solution to the paradox is proposed, which, it is argued, illuminates the truth status of axioms in formal models of programs and Turing machines.

Keywords Gödel's incompleteness · Turing computability · Penrose · Axioms · Consistency

Introduction

It is well known that Lucas (1961), Penrose (1989, 1994), and others have presented arguments, grounded upon Gödel's first incompleteness theorem, to show that human cognition cannot be completely modeled by any computer program or Turing machine. The present paper seeks to show that particular assumptions and strategies employed by Lucas and Penrose lend themselves to the production of an

R. F. Hadley (✉)
School of Computing Science and Cognitive Science Program, Simon Fraser University, Burnaby, BC, Canada V5A 1S6
e-mail: hadley@cs.sfu.ca

apparently serious paradox. In effect, certain of their assumptions and modes of reasoning can be adapted to “prove” that not only are humans not machines, but machines are not machines! Of course, such reasoning must contain some flaw. In what follows I develop the paradox and offer a solution to it.[1] The development of the solution provides insight into the nature of axioms within formal models of Turing machines and computer programs. In particular, the truth status of such axioms is shown to be problematic. In addition, other subtleties relevant to computability and recursion theory are explored in the process.

The paradox in question arises once the feasibility of a particular kind of computer program is established. Before details of this program’s functionality are presented, it may be helpful to consider what prompted the conception of this program. To this end, a brief digression is in order. As noted above, the arguments of Lucas and Penrose rely upon Gödel’s first incompleteness theorem (Gödel 1931). However, it is essential to note that their arguments, as well as those which I offer in the present paper, require only the first half of Gödel’s 1st incompleteness theorem. This portion of the theorem requires only simple consistency of the given formal system. Moreover, there are proofs of the entire theorem (such as Kleene’s 1967 [pp. 248–250], and Boolos and Jeffrey’s 1980) which require just simple consistency. For this reason, we shall not be concerned with ω-consistency in what follows. Simple consistency will suffice. Consequently, we may construe the relevant portion of the theorem as follows: For any consistent, formal number theory, having conventional arithmetic power, there exists some well-formed sentence, true on the standard arithmetic interpretation, and expressible within the theory, but not derivable in it.

Now, the arguments of Lucas and Penrose (among others) differ in some important respects, but they both rely upon (a) particular proven results of recursive function theory, and (b) the same intermediate conclusion. The proven results (a) are these: every Turing machine is input–output (extensionally) equivalent to some recursive function, and in consequence, for each such machine there exists a formal logical system whose deductive closure includes all the machine’s output. The intermediate conclusion (b) sought by these arguments is that no Turing machine, whose output includes many standard theorems of number theory, could ever prove the truth of a gödel-sentence for a consistent, formal deductive system that is equivalent to that very machine.[2] With regard to the writings of Lucas, this intermediate conclusion (b) has been challenged by Lewis (1969) and Webb (1980), both of whom emphasized the existence of an *algorithm* for generating a gödel-sentence for any given, sufficiently strong formal system that satisfies canonical syntactic conditions.

Various discussions have arisen concerning the distinction between *generating* a gödel-sentence and *proving* its truth, but neither Lewis nor Webb noted that, even accepting that distinction, a serious puzzle remains. For, the existence of such a

[1] I will be using ‘paradox’ in its dictionary sense, which does not imply unsolvability.

[2] I use the expression ‘gödel-sentence’ just as the reader would surmise, i.e., to refer to a sentence which is true (on the standard arithmetic interpretation) and representable in the given system, but not derivable in that system.

generation algorithm implies that a Turing machine, M (of the kind described above), could apply that algorithm to its own equivalent formal system, S. This, in turn, appears to entail that the gödel-sentence so generated would be derivable within M's own formal model, i.e., apparently within S itself. Of course, a derivation of a gödel-sentence for S, within S itself, would be impossible if S is consistent (given Gödel's first incompleteness theorem, and given that S is a formal model of a machine of sufficient arithmetic power). But, there are strong reasons to believe that S would be consistent.

In 1985, the above puzzle was noted, discussed and solved by the present author (see Hadley 1987). The solution emerges once we realize that a formal model of machine M, without its input specified, is only a proper subset of a formal model of M *applied to some specified input*, $\mathcal{I}$. This is true even when $\mathcal{I}$ is a formal deductive model of M *simpliciter* (i.e., M without its input).[3] In short, whereas the machine, M, could, by applying the algorithm hitherto mentioned, generate a gödel-sentence for the smaller system, S, that fact provides no reason to suppose that M's output should be derivable within S itself. Rather, such output would be derivable in the larger system that models M-applied-to-its-input.

For some years, I remained content with the solution described above and believed that no *genuine* paradox could arise in connection with Gödel's theorem and Turing machines. Recently, however, I revisited these issues and have concluded that an AI (Artificial Intelligence) program should be able to construct the larger formal system that results (in logical space) when the AI program is presented with a formal model of own functionality.

In what follows, I argue that such an AI program could exist and could generate a gödel-sentence for this larger formal system. As a consequence, this larger formal system would apparently contain a derivation of a gödel-sentence pertaining to itself. This consequence would, of course, violate Gödel's incompleteness results, provided the formal system is adequate for number theory and is consistent. Yet, there are strong reasons to believe that these last two qualifications do hold; hence the puzzle.

As readers familiar with Penrose's arguments (1989, 1994) may soon observe, there are both important differences and partial analogies between his arguments and those presented here. Among other things, Penrose employs a *reductio ad absurdum* strategy to refute an initial supposition that the human mind is essentially mechanistic. In the process, he invokes Gödel's theorem in the derivation of a contradiction needed to complete his *reductio* argument. In contrast, although my argument yields a paradox, grounded upon a derived impossibility, only machines and theorems are required. The fact that an impossibility can be derived even when

[3] In recent personal communication, one logician has casually suggested that this last claim might be false in the case where the formal model (that serves as input) is *inductively presented* to the machine. However, I doubt that this suggestion will bear fruit. No matter how a formal system is presented to a machine, the machine will need to store at least something in its memory. Once this occurs, the machine's memory contents will change, and this must be reflected in any formal model of the changed machine. Moreover, the output of *a machine applied to its argument* would be derivable within a formal model of the machine that includes the specified input, but such output would not be derivable in any formal model of the machine that omits a specification of its argument.

the given agent is unquestionably a machine, following entirely algorithmic processes, strongly suggests that the true difficulties reside at a deeper level than Penrose had supposed.

Concerning this, it is noteworthy that both Penrose (1994) and Lucas (1961) offer arguments that any formal deductive system, which is purportedly equivalent to their respective minds (*qua* supposed machines), would be a *consistent*, sufficiently strong system. They are obliged to do so, of course, because of the restrictions on Gödel's theorem. However, both Lucas and Penrose concede that, over an extended period of time, humans typically exhibit some cognitive inconsistency.

Lucas, in his replies to critics, attempts to meet this difficulty by stressing that, although humans are not entirely consistent in their life-span set of beliefs, they do strive for consistency, and rational humans seek to ensure that their reasoning is valid. Moreover, they attempt to remove inconsistencies from belief sets when these are discovered. From these considerations, Lucas concludes that his purportedly equivalent formal model would be consistent.

Few philosophers have found Lucas convincing on this issue, but Penrose has fared better with his argument. Penrose stresses that it is entirely plausible that logically gifted humans, at least, are capable of perfectly sound reasoning for limited periods of time. In particular, someone such as himself, who understands Gödel's methods of meta-proof, *vis a vis* the incompleteness issues, is capable of valid reasoning for the duration of time required to apply those methods to any formal system which is alleged to be equivalent to the person in question. Because of the sound reasoning exhibited, Penrose contends that, if he is indeed equivalent to some Turing machine, any formal deductive model of that machine, during the relevant time period, would necessarily be consistent.

In partial analogy with Lucas and Penrose, I shall also be presenting a consistency argument for the formal system that is equivalent to the AI program mentioned earlier. However, since the program in question is entirely deterministic, and is assumed to follow only sound methods of inference, the case for the consistency of the formal system will be at least as strong as any argument that Penrose or Lucas can muster for the human case.

The plan for the remainder of the paper is as follows:

The next section presents further background information and a key premise.

The succeeding section describes input specifications and operations performed by a particular AI program in such detail that the possible existence of such a program will be evident.

Following that, I argue that a paradox follows from the existence of the program, when conjoined with standard results of recursive function theory.

The concluding section presents a summary and a probable solution to the paradox, together with a discussion of its implications.

In order to develop the paradox forcefully, I will present several arguments as persuasively as can be, as though I were Lucas or Penrose developing their *reductio* arguments.

Further Preliminaries and a Key Premise

The arguments presented in the remainder of this paper require one premise not commonly found in standard works on computability and logic. However, this premise (labelled $\mathcal{A}$, just below) appears uncontroversial. Certainly, its truth will be readily apparent to researchers who commonly deal with logical models of Turing machines. (N.B., In all the following discussion, I restrict the class of programs and Turing machines under consideration to just those that were explicitly designed to accommodate some previously specified input format. We are *not* concerned with computer "programs" that serendipitously succeed in reading their input.)

($\mathcal{A}$) Let C denote any class of Turing machines which have each been designed to accept input data that conforms to *the same* precise specifications, and let M denote any machine within class C. Further, assume that any given argument (or input set) to be supplied to M will be *axiomatized* according to a fully explicit, canonical method. Then, provided this particular canonical method is known in advance, it is possible to axiomatize the functionality of M itself in such a way that the canonically formulated axioms that represent a specific argument (input) to M will *deductively harmonize* with those axioms that encode M's functionality. By "deductively harmonize", I simply mean that the union of the two axiom sets comprises a formal deductive model of M applied to its argument.

(Readers who are not well versed in axiomatic models of Turing machines are advised at this point to read the brief Appendix A.)

Before delving into details of the following sections, a further preliminary deserves mention. Within the realm of computability, the concepts of *primitive recursive function* and *general recursive function* play a crucial role. (Readers already familiar with these concepts may skip to the following section.) For present purposes, we may construe primitive recursive functions as those which (a) are expressible as (or are equivalent to) functions representable within standard (high school) algebra, and (b) whose evaluation can be computed in a finite span of time. In contrast, the class of *general* recursive functions, though it includes all primitive recursive ones, also includes functions whose computation is non-terminating in the following way: the computation process generates a *countable* infinity of values (such as the squares of all positive integers). Those general recursive functions which are not primitive recursive are still defined, in large part, in terms of sub-expressions which represent primitive recursive functions.

Assumptions and Specifications Concerning a Particular AI Program, P

Let us assume that a program P has in its memory a set of axioms, encoded in first-order logic with equality, which are equivalent to the seven number-theoretic axioms of the system known as Q. (I am adopting the version of Q presented by Boolos and Jeffrey 1980.) P is also programmed to apply standard rules of inference, from first-order predicate calculus (with equality), to these axioms, to enable P to derive theorems from the axioms. For convenience, I shall refer to the entire system (the Q axioms together with the first-order logic in which they are represented) as Q. Also, for simplicity, assume that the first-order logic in question is a natural deduction system. Thus, Q will contain no axioms other than the seven Q axioms previously mentioned. Note that the entire Q system is adequate for deriving an enumerable subset of the theorems of arithmetic. Recall also that although Q is, in some respects, a weak system, it is strong enough to be subject to Gödel's incompleteness theorems. Moreover, it is strong enough to represent any given recursive function (Boolos and Jeffrey 1980). All formal deductive systems which *contain* a deductive system equivalent to Q are here dubbed SSD systems (for "sufficiently strong deductive" systems).

Let us assume that P is programmed so that its *normal* activity is to effectively enumerate theorems derivable within Q. However, prior to assuming its normal activity, P performs a special task, but only once. In particular, prior to the commencement of any processing by P, *special input data* is placed in P's input devices. P is designed so that, at the moment P becomes active, it fetches this special data from the input devices, and performs a distinguished task with this data, which includes generating a sentence and printing it. (This task is elaborated below.) *Upon completion of that task, P assumes its normal theorem proving activity*. The presence of this normal activity is a crucial aspect; readers are asked to bear in mind that, in its entirety, P's behaviour is non-terminating. Note also that, because P's normal task is to enumerate theorems of Q, any formal system equivalent to P must be an SSD system.

Program P is designed on the assumption that its input is accurate and is presented in a canonical form. In particular, a correct input set (or argument) for P is *an SSD system of P itself*, and is to be presented as a finite series of character strings, where each string is separated by three consecutive semicolons. The *character strings* are read by P in a sequential fashion, one character at a time. Each character string corresponds to one axiom or inference rule of the system being presented. As P reads each successive character, P places the character in an axiom of the form, *character(n, p, c)*, where "n" denotes the number of the axiom or rule being presented, "p" denotes the position of a character within that axiom or rule, and "c" denotes a given character within the axiom or rule numbered "n". (Clearly, it is a simple matter to ensure that P keeps track of which axiom and position it is currently dealing with.) In addition, as P reads the various character strings, P concatenates the genuine "data characters" (while discarding the separator semicolons), and in so doing P constructs and stores all the axioms and rules of the given SSD system. Each axiom and rule so constructed will only be *treated as data* in the form of character strings. Furthermore, we should *not* assume P assigns any semantic content to the axioms and rules.

Apart from the above, two further prescriptions upon P's input are stipulated. Please recall that I have assumed the truth of premise $\mathcal{A}$, explained at the outset. (The reader is advised to read that premise once more.) In light of $\mathcal{A}$, the first of the following prescriptions is justified. The prescriptions are:

(i) That the given SSD system, S (supplied as P's argument), has been canonically created so that, when S is *axiomatized as character data* by P (which we know will result in a set of axioms of the form *character(n, p, c)*) the union of the resulting set of axioms with S itself will be *deductively harmonized* in the special sense defined earlier. Bear in mind also that S is required to be an SSD model of P itself.

(ii) That the given SSD system is finitely represented within Q.

Regarding condition (ii), detailed arguments are presented in the next section to establish that program P's own equivalent formal system is finitely representable in Q. As a brief preview, let us recall that P's normal activity (enumerating theorems of Q) ensures the SSD property, and P's status as a functioning program entails P's equivalence to some recursive function. Moreover, every recursive function is finitely representable in Q (for more on recursive functions and their relation to Q, see Boolos and Jeffrey, Chapter 14 1980). Finally, given well known theorems of Kleene (1936) and Turing (1937), we safely infer that each such recursive function is equivalent to some Turing machine.

Having described P's input requirements, we may now further specify and summarize program P's functionality. In particular, let us assume that when P is presented with an input stream, P takes the following actions:

1. As P reads the SSD data, P places each "data character" into an axiom of the form, *character(n, p, c)*, and then stores the axiom in its memory. In doing this, P is *axiomatizing its own input data*. We could equally well assume that P axiomatizes this input data according to whatever canonical scheme is deemed necessary. Note also that the axioms being constructed may be regarded as true, since we assume that P follows a correct method.
2. While performing step (1), P interleaves another process in which it progressively assembles all the axioms and rules of the given SSD system. It merely concatenates the characters it reads in order to do this, while observing the ";;;" separators between the intended character strings. P saves all the axioms and rules that it constructs in a region of memory reserved for string data. We may call this collection of axioms and rules, system S_0. Recall that S_0 is the formal equivalent of a Turing machine and, by virtue of satisfying P's input specifications, S_0 is already represented within the system Q.
3. P takes the union of the axioms created in step (1) and the system S_0. Call this union S_1. Note that S_1 is an SSD system. Moreover, because (a) S_0 is represented within Q, (b) the axioms created in step (1) all involve the same predicate (namely, 'character'), and (c) all arguments to this predicate are simply numbers, the system S_1 will itself be finitely represented within Q, provided the language of Q is extended to include a finite number of predicate names, including 'character'.
4. We assume P has also been programmed to apply a general algorithm (effective function), such as that given by Feferman (1960), for generating a specific (tailor-made) gödel-sentence for *any given* SSD system that satisfies a predetermined canonical description.[4] The applicability of Feferman's

[4] Feferman's algorithm is a primitive recursive function. Thus, it is certainly Turing computable and programmable in any standard computer language.

generalized method to S_1 is demonstrated in the following section. (This applicability entails, among other things, that it would be utterly pointless, within this paper, to actually arithmeticize S_1, or to produce its proof predicate.) Assume that P now applies this algorithm to the system S_1, thereby generating a sentence, which P then prints.

5. After printing the gödel-sentence, *P deletes from its memory any information added during the execution of steps 1– 4, inclusive*. Following that, P enters its normal "theorem-proving" mode.

Now, note that if P has been given appropriate and canonically organized input data, and if the SSD system it has been given contains only true axioms and valid inference rules, then P will have generated a true gödel-sentence, and that sentence will be P's output. In general, whenever P is given correct and true information, P generates true output. Moreover, by hypothesis, P follows only correct algorithms for creating the initial set of axioms that axiomatize the character information it has received belonging to the SSD system it has been given.

Demonstration of Paradox

The Language of Program P

To facilitate arguments presented below, and for concreteness, let us stipulate that P is written in *pure LISP*, which is a computationally sufficient (in Turing power) subset of the LISP programming language.[5] Many programs have been written in pure LISP, and all such programs share this essential feature, viz., they all possess a purely functional, hierarchical structure. At the top of each such hierarchy, will be a master function, which invokes subordinate functions, which may in turn invoke yet more subordinate functions. For example, suppose that *f1, f2, f3, f4* and *f5* are function names, and that *x, y* and *z* are variable names. Then (ignoring minor notational differences) the expression

$$f1\,(f2(\,(f3\,x),\;(f4\,y)\,),\;(f5\,x\,z)\,)$$

exhibits a functional structure (where *f1* is the "master function") that exemplifies the manner in which a pure LISP program would be executed. It is noteworthy that both the functional structure of pure LISP, and its manner of defining functions were closely based upon Church's λ-calculus (Church 1936). Indeed, to each program written in pure LISP there corresponds a precisely equivalent formula in the λ-calculus (Fisher 1993). Typically, such a formula would involve deeply nested sub-expressions.

P's Equivalence to a Recursive Function

Among academic computer scientists, there is a strong consensus that corresponding to every working computer program, there exists a Turing machine whose input–

[5] It is straightforward to encode any Turing machine table in pure LISP.

output behaviour matches the given program (cf. Turing 1936–1937). However, the reasons behind this consensus are often not articulated, and some readers may question whether the program P, which is capable of performing two distinct functions, could actually possess a Turing equivalent formal model. After all, Turing machines typically compute just a single function, whereas P can switch from generating a gödel-sentence to operating as a standard theorem prover in first-order logic. Let us pause, therefore, to consider just why it is certain that P is equivalent to some Turing machine and to some general recursive function.

Recall that we have stipulated that P is entirely written in pure LISP (and note that the "master function" in a pure LISP program may perfectly well invoke subfunctions which have entirely separate roles). We have also seen that each program written in pure LISP is represented by a functional expression (usually deeply nested) whose structure is fully equivalent to a formula definable within Church's λ-calculus. This entails that each such program is a λ-definable function. Other sufficient arguments could be given, but this λ-definability property, in conjunction with well established theorems of recursion theory, fully entails P's equivalence both to a general recursive function and to some Turing machine. For, Kleene (1935, 1936) proved the equivalence of λ-definability and general recursiveness, and Turing (1937) proved that for each λ-definable function there is an equivalent Turing machine.

Furthermore, as previously noted, every general recursive function is finitely axiomatizable in Q (proven in Boolos and Jeffrey 1980). Thus, it is certain that P has a corresponding formal deductive model, representable in Q.

The Consistency of P's Formal Model

Concerning the existence of P's equivalent formal deductive systems, please recall that each such system must, among other things, contain axioms or inference rules that correspond to P's capacity to apply an algorithm (such as Feferman's 1960) for generating gödel-sentences. To simplify the following argument, and to quell any doubts about "exotic inference rules", we shall henceforth be concerned only with those formal models in which all of P's processing capacity is captured by means of *axioms* being added to the system Q. This will always be possible, because any *special* inference rules that might appear necessary can be recast as axioms having a conditional form. Moreover, it is essential to note that, relative to the standard interpretation of arithmetic, Feferman's algorithm is truth-preserving. Given a true arithmetic theory of adequate power, the algorithm generates a true sentence.

Now, recall that P has in its memory the axioms of Q, and that P is assumed to follow sound, standard rules for generating proofs within system Q. We assume, as is common practice, that the axioms of Q are true. Given that P has been constructed to follow only correct algorithms, that it employs the valid inference rules of Q, and that *the axioms within its memory will all be true when P is given accurate information*, it follows that any correct formal deductive model of P, immediately *after* P has been supplied with valid input data, can generate no contradiction. That is, such a deductive model of P must be at least simply consistent. Moreover, since P normally spends its time enumerating theorems within Q, and no upper bound can be placed on the number of theorems so generated, any formal model of P itself (in conjunction with its true input data) must be an SSD system—a *consistent* SSD system in fact. (Recall that simple consistency, rather than ω-consistency, is sufficient for the portion of Gödel's theorem which concerns us.)

Having established that P does possess some consistent SSD models, we may rightly infer, via the well-known Löwenheim-Skolem theorem (1915), that each of these consistent models is satisfiable within the domain of the natural numbers. (The relevance of this point will emerge presently.)

In addition, because P (without its input supplied) is equivalent to some recursive function, we are assured that P's equivalent SSD system (hitherto named S_0) is finitely representable within Q. Let us assume that it has been so represented. Then, we may deduce that this SSD system can be constructed in the canonical fashion that satisfies P's input requirements. The latter follows as a consequence of two facts. First, premise $\mathcal{A}$ assures us that P's SSD system can be formulated in a fashion that satisfies condition (i) of P's input requirements (i.e., the relevant axiom sets can be deductively harmonized). Secondly, condition (ii) is satisfied because P's own SSD system is representable within Q.

Having seen that P's equivalent SSD system can satisfy P's own input specifications, we shall assume that this system is actually formulated to ensure that satisfaction. Call this system S_{P}. Note that, for the intended domain of interpretation, S_{P} contains only true axioms and valid inference rules, because S_{P} embodies only the system Q and P's algorithmically correct functionality. (I am assuming that, wherever applicable, the non-logical terms of S_{P} are given their standard number-theoretic interpretation.) Note also that, because P *uses* character axioms (which P has created from input strings) as a basis for some of its subsequent processing, the system S_{P} will contain axioms that relate this functionality of P to the character axioms themselves. This fact, by itself, largely ensures that S_{P} can be deductively harmonized with the character axioms.

The Paradox

Now, program P is not an especially difficult AI program, and it is certainly a logical possibility that some logician would formulate S_{P} and present it to P. Therefore, let us consider what would happen when P is presented with S_{P}. From step 1 (in P's functional specification), we know that P will create a new set of axioms which describe the characters occurring in S_{P}, and the positions they occupy in each character string. These new axioms may be taken as true, as noted earlier.

From step 2, we learn that P will also create the axioms and rules of the SSD system that comprises S_{P}, and that P will save these axioms and rules in its memory.

Step 3 tells us that P forms the union of the "character axioms" created in step 1 with the axioms and rules of S_{P}. Because of the *input specifications* satisfied by S_{P}, and the canonical nature of the character axioms just mentioned, the system that results from this union will be "deductively harmonized". Call the resulting union $S_{\mathrm{P+input}}$. Note that this new system will be an SSD model of P applied to the argument that P has just received. This is so because the union set includes both the SSD system that captures P's recursive functionality and the axioms that describe the data that P has just processed. Recall also that $S_{\mathrm{P+input}}$ is consistent and will already be represented within Q, provided (as we assume) the language of Q has been extended to include predicate names such 'character' (see step 3 of the section entitled "Assumptions and Specifications Concerning a Particular AI Program, P").

At this stage, program P is ready to employ Feferman's algorithm, and a brief digression will now establish that this algorithm is applicable to $S_{\mathrm{P+input}}$. For, as Feferman (1960) makes clear, his effective procedure is applicable to any sufficiently strong (in the SSD sense), first-order formal system that is finitely axiomatizable. We have seen that $S_{\mathrm{P+input}}$ does possess these attributes; it is finitely axiomatizable in the first-order system Q, and it is sufficiently strong in the relevant sense. To be sure, Feferman's proofs also assumed, for convenience, that all variables of the given formal system would range over the natural numbers. However, this aspect is not a problem. For, we have seen, due to $S_{\mathrm{P+input}}$'s consistency and the Löwenheim-Skolem theorem, that $S_{\mathrm{P+input}}$ is satisfiable in the domain of the natural numbers. Given this fact, I will hereafter assume that $S_{\mathrm{P+input}}$ has been assigned an interpretation that satisfies all of $S_{\mathrm{P+input}}$'s axioms within that numerical domain. In consequence, the range of $S_{\mathrm{P+input}}$'s variables is no longer an issue, and the conditions for the applicability of Feferman's algorithm are all clearly satisfied. Note well, however, that a change in the semantic interpretation assigned to $S_{\mathrm{P+input}}$ does *not* alter its deductive closure. Derivability in a formal system is, of course, a purely syntactic matter. For this reason, $S_{\mathrm{P+input}}$ remains a valid SSD equivalent to the program P. (Each sentence generated by P must still reside in $S_{\mathrm{P+input}}$'s deductive closure.)

Returning now to the main argument, according to step 4, P now applies Feferman's algorithm (for generating a gödel-sentence) to system $S_{\mathrm{P+input}}$, and prints the resulting sentence. Call that sentence G. Because G is the output of P applied to P's argument, and because P is a recursive function, G must be derivable in any *functionally-equivalent* formal deductive system that corresponds to P applied to its argument. (This last claim is elaborated presently.) But one such deductive system is just $S_{\mathrm{P+input}}$, which is the system that P constructed. Moreover, that system (which we know to be an SSD system) is consistent. Thus, a gödel-sentence for at least one consistent SSD system must be derivable within that very system. As we know from Gödel's first incompleteness theorem, this is impossible. Thus, from the logical possibility that P could be presented with its own SSD system, we have derived an impossibility. This, of course, is the paradox.

One point in the preceding paragraph bears elaboration. It is crucial to realize that the gödel-sentence, G, which P writes on its output device, appears in the same fashion as the various "Q theorems" that P outputs. *All* sentences which P prints are well-formed formulae of Q, and collectively they comprise the recursively enumerable set which is the value of P-applied-to-its-argument. Because $S_{\mathrm{P+input}}$ does capture the recursive functionality of P, each of the sentences within that recursively enumerable set will be derivable within $S_{\mathrm{P+input}}$, or some equivalent sentence will be so derivable. ($S_{\mathrm{P+input}}$ is *not* a predictive theory that describes P's *actions*, rather $S_{\mathrm{P+input}}$ *expresses* the recursive function that P realizes.)

Discussion and Conclusion

I hasten to stress that the foregoing argument is *not* intended as a *reductio* against Gödel's first incompleteness theorem. There is not the slightest doubt that his proof is correct. It has been examined countless times by competent logicians, and equivalent results have, in the intervening years, been proven by independent methods.

Apart from Gödel's first incompleteness theorem, I have assumed both the correctness of Turing's (1937) proof that Turing computability is equivalent to λ-definability and Kleene's (1936) proof that the class of λ-definable functions is equivalent to the class of general recursive functions. Yet, these results of Turing and Kleene are not at all contientious. As a last resort, they might be questioned, but as will emerge, that appears unnecessary.

Perhaps some readers will, in light of the derived paradox, argue for the rejection of my initial premise (labelled $\mathcal{A}$), which asserts that there will always exist *some* axiomatization of a Turing machine's functionality that can be deductively harmonized with a prior axiomatization of any legal argument to that machine. Yet, this premise merely expresses what is common knowledge among researchers in automata theory. In addition, a concise and strong justification was offered for premise $\mathcal{A}$ in Appendix A.

Apart from the above, my argument assumed the existence of an algorithm, such as Feferman's (1960), for generating gödel-sentences. Yet, the existence of such algorithms has been rigorously proven, not only by Feferman, but by Smullyan (1994). As hitherto noted, the existence of these algorithms is widely accepted, and has been cited by Lewis (1969) and Webb (1980), among others. Nevertheless, it is not preposterous to suppose that contrary to published results, there can be no absolutely general algorithm for generating a gödel-sentence for each possible SSD system. We know that there can be no general algorithm for ascertaining whether each possible Turing machine will halt on a given input. Conceivably, a somewhat analogous situation holds with respect to apparent effective procedures for generating gödel-sentences.

However, there remains a more credible solution to the paradox than the rejection of such generation algorithms. For a necessary step in the development of the paradox was to establish the consistency of the system, $S_{\mathrm{P+input}}$, which models the program P applied to an SSD model of itself. The argument for the consistency of this system resembled that of Penrose (and to a degree, that of Lucas) in that it assumed that axioms which represent sound reasoning capacities (and perfectly rational capacities such as the ability to concatenate a string of characters and store them in memory) would be true axioms, or at least be mutually consistent. Moreover, the consistency argument does possess considerable plausibility, for the program P would yield entirely valid results if it were given, as input, any consistent, first-order SSD system other than its own! Recall that the derivation of the paradox crucially relied on P's input being an SSD model of itself. This fact strongly suggests that P's own rationality is flawless.

Note also that P's algorithmic capacities are all individually reasonable. They amount to these: the ability to derive theorems of Q via universally sound rules of first-order logic; the ability to apply Feferman's algorithm for generating gödel-sentences; the ability to create manifestly true axioms of the form of *character(n, p, c)*; the ability to form the union of two sets; and the ability to concatenate characters and to store the resulting strings in memory. (By the way, these are all abilities which are possessed by Penrose himself.) Moreover, no difficulties arise in supposing that an SSD model of P *simpliciter* (without input) is consistent. Indeed, I remain convinced that *that* system, S_P, is consistent.

However, a fallacy arises when we suppose that a set of axioms which truthfully describe P's input (i.e., the set of axioms of the form, *character(n, p, c)*) will necessarily be consistent with S_P. To be sure, no matter which SSD system is supplied to P as input, the resulting set of character axioms will contain only true sentences. Also, it seems reasonable to assume that all axioms in the consistent system, S_P, will be true, since program P can execute properly. Thus, it would seem that the union of the true character axioms with the true axioms in S_P must form a consistent set as well. (That union is just $S_{P+input}$, all of whose inference rules are sound.) However, the error is this: many of the axioms in S_P encode algorithmic processes rather than truths of arithmetic. Although it is tempting to suppose that such axioms are *true*, the reality is that they do not always describe states of affairs.

For example, within an axiom set that encodes Feferman's algorithm, there will be axioms that represent a method for calculating gödel-numbers of complete sentences. The choice of a function for assigning gödel-numbers to sentences is to a certain degree arbitrary, and axioms that represent the evaluation of that function for particular arguments are closer to being *instructions* than descriptions of facts. Moreover, although many axioms in S_P will have the form of conditionals, this does not entail that they have a truth value. To be sure, the conditionals can possess meaning, just as conditional instructions in modern programming languages have semantic interpretations. However, the possession of meaning does not guarantee a truth value. (This is related to the well known declarative vs. procedural distinction.) Consider a set of if-then instructions which merely encode an algorithm for concatenating the first, third, and fifth letters in any eight letter word. The algorithm is an arbitrary process, and axioms which encode that process, say in the Prolog programming language, are essentially stipulations. It is tempting to suppose that sentential stipulations should always be regarded as true, but the temptation is misleading. (Consider a Prolog program that proved theorems of Q arithmetic, but also printed the negation of each theorem it derived.)

In light of the above points, we can understand how a true set of character axioms, that encode graphical aspects of S_P, could still be inconsistent with S_P itself. The totality of axioms within S_P, although not false, may yet possess the appropriate formal properties to enable a false gödel-sentence to be derived within the union of S_P and the true character axioms. The character axioms, although true, enable axioms of S_P to operate upon their own syntax and vocabulary, and that is where the danger lies. There is no necessity for the expanded system, $S_{P+input}$, to be consistent, and the existence of the paradox strongly suggests that it is not.

The lesson of these points for Lucas and Penrose will, of course, be moderately obvious. However sound the reasoning capacities of their minds (or brains), and however consistent a formal model of these capacities may be, there is no assurance that the union of this model with *true* axioms that describe the strings of that same SSD system will itself be consistent.

Acknowledgements I am very grateful to Wilfried Sieg for his penetrating comments on a precursor of this paper, and to Alistair Lachlan and Warren Burton for some technical remarks. I also wish to thank Eugenia Ternovska for her helpful comments on degenerate cases of recursive functions, and David Mitchell for a stimulating theoretical discussion. Thanks also to Mark Stickel for useful comments on the properties of the automated theorem provers, PTTP and OTTER and to Amin Sharifi for initial references to those systems.

Appendix A

For those not accustomed to dealing with axiomatic models of Turing machines, the following remarks should be helpful. It is widely recognized within the field of computer science that programs presented in modern programming languages can always be written so as to accommodate a previously specified input format. Moreover, it is common knowledge among researchers in the *logic programming community* that, corresponding to any working computer program, there is an equivalent program written in *Prolog* (which is the most widely used, logic programming language). Prolog, like all other modern high-level programming languages has the computational power of a universal Turing machine.

Now, any Prolog program consists of a series of axioms in the form of horn-clauses, most of which are syntactic variants of axioms written in a first-order predicate calculus. When input data is to be supplied to a Prolog program, it must first be converted, either by humans or by supplementary program code, into axioms that have the form of atomic sentences. In order for the Prolog program to be successful, it must be designed in a fashion that is adapted to some previously determined, axiomatic representation of the input data. Happily, this is always possible, because if need be, it is always possible to augment the main program's axioms with special axioms that perform a *bridging function*, so that the entire axiom set, including the "input data axioms", is deductively harmonized.

Now, admittedly, the axioms of Prolog programs do not always perfectly correspond to sentences written in classical first-order logic. However, there are programming environments, based upon first-order theorem provers, which *do permit* the required horn-clause axioms to be written entirely within classical first-order logic. Examples of such programming environments are Otter and PTTP (Prolog Technology Theorem Prover), both of which include powerful first-order theorem provers.[6] Otter, in particular, is a very powerful, inferentially complete theorem prover that permits the full range of predicate calculus syntax to be employed in the programs it executes. A remarkable characteristic of the programs which Otter and PTTP accept is that the axiom set that comprises each such program can serve as the program's own *equivalent formal deductive model*.

[6] Documentation for the Otter and PTTP systems can be found, respectively, at these websites: http://www−unix.mcs.anl.gov/AR/otter/description and http://www.ai.sri.com/∼stickel/pttp.html

It is also noteworthy that, for any Turing machine, there exists a corresponding set of first-order axioms that is functionally equivalent to that machine (see Davis, Chapters 1 and 6, 1958, for a discussion of Post's method, whereby any Turing machine can be presented axiomatically). Moreover, the axiom sets that constitute such programs can be designed to deductively harmonize with their "input data axioms" in a fashion that is strongly analogous to the fashion in which an actual Turing machine is designed to accommodate a predetermined presentation of input data upon its machine tape.

References

Boolos, G. S., & Jeffrey, R. C. (1980). *Computability and Logic*, (2nd ed.). Cambridge, UK: Cambridge University Press.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics, 58*, 345–363.

Davis, M. (1958). *Computability and unsolvability*. New York: McGraw-Hill Book Company.

Feferman, S. (1960). Arithmetization of metamathematics in a general setting. *Fundamenta Mathematicae, 49*, 35–92.

Fisher, M. J. (1993). Lambda-calculus schemata. *Lisp and Symbolic Computation, 6*, 259–288.

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

Hadley, R. F. (1987). Gödel, Lucas, and mechanical models of the mind. *Computational Intelligence, 3*, 57–63.

Kleene, S. C. (1935). λ-definability and recursiveness. *Bulletin of the American Mathematical Society, 41*, 490.

Kleene, S. C. (1936). λ-definability and recursiveness. *Duke Mathematical Journal, 2*, 340–353.

Kleene, S. C. (1967). *Mathematical logic*. New York: John Wiley & Sons, Inc.

Lewis, D. (1969). Lucas against mechanism. *Philosophy, 44*, 231–233.

Löwenheim, L. (1915). Über Möglichkeiten im Relativkalkül. *Math.Ann., 76*, 447–470.

Lucas, J. R. (1961). Minds, machines, and gödel. *Philosophy, 36*, 112–117.

Penrose, R. (1989). *The emperor's new mind*. Oxford: Oxford University Press.

Penrose, R. (1994). *Shadows of the mind*. Oxford: Oxford University Press.

Smullyan, R. M. (1994). *Diagonalization and self-reference*. Oxford: Clarendon Press.

Turing, A. M. (1936–1937). On computable numbers with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*. 42, (pp. 230–265). A correction is given in Vol. 43, pp. 544–546.

Turing, A. M. (1937). Computability and λ-definability. *Journal of Symbolic Logic, 2*, 153–163.

Webb, J. (1980). *Mechanism, mentalism, and metamathematics*. Hingham, MA: Reidel.