AN ERROR IN A COMPUTER VERIFIED PROOF OF INCOMPLETENESS BY NATARAJAN SHANKAR

James R Meyer

http://www.jamesrmeyer.com

v1 03 November 2011

Abstract

This paper examines a proof of incompleteness by Natarajan Shankar, who claims that his proof has been verified by computer. This paper demonstrates that the proof has a fundamental error of logic which renders the proof invalid.

1 Introduction

This paper demonstrates a fundamental error in a proof of incompleteness by Natarajan Shankar, published in 1994 [1], though Shankar first coded it in 1986. This paper does not discuss the implications of the failure of the computer verification software to detect the error in the proof; that will be published elsewhere.

2 Shankar's proof

Shankar's description of his incompleteness proof can be found in Shankar's book on meta-mathematics [1]. It is written using the Boyer-Moore verification system [2], which uses the Lisp programming language[a]. Shankar's proof is a proof of incompleteness of a single formal system; the formal system is a system called Z2.[b] The Lisp language system does not directly use the actual symbols of the Z2 formal system. Instead certain Lisp expressions are used which correspond to the formulas and terms of the Z2 system, and which are called *z−expressions*. For example, the *z−expression* `(P 1 2)` represents the *'is an element of'* symbol $\in$.

In Lisp a function is written as `(fn` $\mathtt{a}_i$ `...`$\mathtt{a}_n$`)`, where `fn` is a function name, and $\mathtt{a}_i$ are Lisp expressions. Unlike many programming languages, Lisp has no data typing; data is inputted as either:

- elemental data, which can be numerals, or symbols or symbol combinations. This includes the data T and F, for Boolean values 'true' and 'false'.

[a] The system referred to by Shankar has now been superseded by ACL2, see [3]

[b] The details of the Z2 system are immaterial here; they are given in Shankar's book.

- list data, where the items in the list can be either lists or element values. In the Lisp of Shankar's proof, all lists are constructed by the use of ordered pairs. Note that, a function that returns data can also be an item on a list.

Shankar mentions various divisions of Lisp expressions. The first division is into either syntactical expressions or data expressions. The term *syntactical expressions* is intended to refer to Lisp expressions that are those that are used to construct Lisp programs, while the term *data expressions* is intended to refer to Lisp expressions that are used as inputs to Lisp programs. However, in Lisp, a syntactical expression can also be used as a data input, so that the same symbol combination that is a Lisp expression can be both a Lisp data expression and a Lisp syntactical expression. Furthermore, every syntactical Lisp expression, when values are given for all its arguments, evaluates to a data expression; and there can be data expressions within syntactical expressions. This should be borne in mind when considering references to such expressions.

Shankar's computer code Lisp expressions are presented here in monospace typewriter font, for example, `(SUBST EXP VAR TERM FLG)`. Some of the other terminology used by Shankar includes:

- The Lisp expression `(NCODE Z)` is a Lisp function which is a Gödel numbering function, which for a given *z–expression* Z, returns a numerical data value. Shankar also gives the alternative notation ⌜ Z ⌝ for this Lisp expression.[c]

- $[a/x,\ b/y]\ \boldsymbol{A}$ represents the Z2 formula that results from the substitution of the free variable x in the Z2 formula $\boldsymbol{A}$ by the value a, and the substitution of the free variable y in $\boldsymbol{A}$ by the value b.[d]

- $\vdash\ \boldsymbol{A}$ represents the concept *'the Z2 formula* $\boldsymbol{A}$ *is provable in the system Z2'* (for a given set of axioms).

Shankar's proof uses a proposition which he calls the Representability theorem[e]. The theorem, as Shankar states it is:
For any unary Lisp function g,
there is a Z2 formula $\boldsymbol{A}_g$ *with two free variables, x,y such that:*

$$\textit{If } \texttt{(g X) = Y}, \textit{ then } \vdash\ [\ulcorner\texttt{X}\urcorner/x,\ulcorner\texttt{Y}\urcorner/y]\ \boldsymbol{A}_g$$
$$\textit{If } \texttt{(g X)} \neq \texttt{Y}, \textit{ then } \vdash\ [\ulcorner\texttt{X}\urcorner/x,\ulcorner\texttt{Y}\urcorner/y]\ \neg\boldsymbol{A}_g$$

Shankar points out the need to provide a rigorous proof of his Representability theorem, stating:

> *'Informal presentations of the representability proof show the details of the representability of one or two meta-theoretic functions and argue that the other functions can be similarly represented. Such an argument is acceptable for an informal exposition where the reader can be urged to grasp the general pattern, but is unacceptable for a formalized proof.'*

[c] See Shankar's section 4.3.2 *Mapping N-Expressions to Sets*
[d] See Shankar's section 1.2.5 *The Representability Theorem*
[e] See Shankar's section 4.4 *The Representability Theorem*

Given the statement of the Representability theorem, it does not appear that there should be any great difficulty in stating the proposition in a fully formal format, where all the implicit assumptions are explicitly expressed. But instead of providing his Representability theorem in a fully formal format, and giving a formal proof of that theorem, Shankar circumvents the need to provide a computer verified proof of the theorem and justifies this by a series of vague informal assertions regarding what he calls a Lisp 'interpreter' function `EV`. He then subjects certain assertions regarding this Lisp function to computer verification. He states:

> *'The proof here has been formalized in the metatheory itself. The representability argument is simplified by demonstrating the representability of a Lisp interpreter, . The representability theorem is just the representability of this Lisp interpreter. It is shown that each of the Lisp functions used in the metatheoretic description of Z2 can be evaluated using the Lisp interpreter. The computability of the metatheory by the interpreter and the representability of the interpreter taken together, yield the representability of the metatheory. This subtle and seemingly innocuous use of a Lisp interpreter is perhaps the single most important "design decision" in the mechanical verification'.*

The result is that, although Shankar claims that he has created a computer verified proof, in fact he has created a proof in which many of the steps of proof are indeed subjected to computer verification, but not every step in the proof is computer verified.

The statement of Shankar's incompleteness theorem is:[f]

> *Where U is* `(SUBST (G GIVEN) 6 (NCODE (GCODE (G GIVEN))) 0)`,
> *then if either* $\vdash U$ *or* $\vdash \neg U$, *then both* $\vdash U$ *and* $\vdash \neg U$.

In generating this 'undecidable sentence' U, Shankar defines a Lisp predicate `P` by:[g]

`(P A GIVEN)` $\stackrel{\Delta}{=}$ `(THEOREM(SUBST A 6 (NCODE(GCODE A)) 0) GIVEN)`

In the above, `A` is a variable with the domain of Lisp $z-$***expressions***. The meaning is that, if `A` is the Lisp representation of $\boldsymbol{A}$, where $\boldsymbol{A}$ is a variable whose domain is the Z2 formulas, then the above asserts that the Z2 formula, that is the result of the substitution of the free variable of a Z2 formula $\boldsymbol{A}$ by the value $\ulcorner$`(NCODE (GCODE A)`$\urcorner$, is provable according to the given axioms (here represented by the term `GIVEN`). The function `SUBST` represents the substitution, where the free variable y of a Z2 formula $\boldsymbol{A}$ is represented here by the number `6`.

So that if `A` is the Lisp representation of a Z2 formula $\boldsymbol{A}$, the argument is that if `(P A GIVEN) = T` (here `T` represents 'true'), then the Z2 formula $[\ulcorner$`(GCODE A)`$\urcorner/y]\ \boldsymbol{A}$ is provable in the given formal system. Shankar asserts that his Lisp evaluator function `EV`

[f] see Shankar's Theorem 5.3.2

[g] see Shankar's Section 5.3, *Construction of the Undecidable Sentence*. Note that the code `THEOREM` is not actually used in the computer proof, but is an abbreviation of the actual code used. This does not affect the principles involved.

gives a representation of this Z2 formula as:

$$\boldsymbol{A}\begin{cases}\ulcorner\texttt{0}\urcorner \\ \ulcorner\texttt{(LIST f-p 0 1)}\urcorner \\ \langle\ \ulcorner\texttt{(GCODE A)}\urcorner,\ \ulcorner\texttt{(GCODE GIVEN))}\urcorner\ \rangle \\ \ulcorner\texttt{(FA)}\urcorner \\ \ulcorner\texttt{(VAL)}\urcorner\end{cases}$$

Shankar states that this Z2 formula has no free Z2 variables, and is therefore a sentence.[h] But while Shankar may have coded it as a sentence, by the definition of the evaluator function `EV`, its second argument is the Lisp expression to be evaluated, which, in this case, is the Lisp expression that corresponds to a variable $\boldsymbol{A}$ with the domain of Z2 formulas. Clearly, this Lisp expression *should* be a variable value, not a fixed value, since it is directly dependent on the value of the Z2 formula $\boldsymbol{A}$.

Shankar now defines a Z2 formula $\boldsymbol{G}$ by substituting $\ulcorner$`(GCODE A)`$\urcorner$ in the above formula by y, which gives the Z2 formula $\boldsymbol{G}$:

$$\boldsymbol{G}\begin{cases}\ulcorner\texttt{0}\urcorner \\ \ulcorner\texttt{(LIST f-p 0 1)}\urcorner \\ \langle\ y,\ \ulcorner\texttt{(GCODE GIVEN))}\urcorner\ \rangle \\ \ulcorner\texttt{(FA)}\urcorner \\ \ulcorner\texttt{(VAL)}\urcorner\end{cases}$$

Shankar claims that this Z2 formula has one free variable y. He says that for this Z2 formula $\boldsymbol{G}$, there is a corresponding Lisp representation, which he calls `(G GIVEN)`. His supposedly 'undecidable sentence' is that which is obtained by substituting the free variable y in the formula $\boldsymbol{G}$ by $\ulcorner$`(GCODE GIVEN))`$\urcorner$. The Lisp representation of this defines the 'undecidable sentence' as:

```
U = (SUBST (G GIVEN) 6 (NCODE (GCODE (G GIVEN))) 0).
```

which gives the 'undecidable' Z2 formula as:

$$\boldsymbol{G}\begin{cases}\ulcorner\texttt{0}\urcorner \\ \ulcorner\texttt{(LIST f-p 0 1)}\urcorner \\ \langle\ \ulcorner\texttt{(GCODE G GIVEN)}\urcorner,\ \ulcorner\texttt{(GCODE GIVEN))}\urcorner\ \rangle \\ \ulcorner\texttt{(FA)}\urcorner \\ \ulcorner\texttt{(VAL)}\urcorner\end{cases}$$

The error in Shankar's argument is now readily apparent, since this 'sentence' is defined in terms of $\ulcorner$`(LIST f-p 0 1)`$\urcorner$, which *should* be a variable value with the domain of terms that correspond to Z2 formulas. Since the 'undecidable sentence' is defined in terms of a value which should be a variable value, that means that this 'undecidable sentence', if it were correctly coded, would have no determinate value until a value is assigned for that variable. Hence the definition of the so-called 'undecidable sentence' does not in fact

[h] Here a 'sentence' means a proposition, so that it is a statement with no free variables

define a single Z2 formula, but a set of Z2 formulas, where the value of each individual formula is determined by the value assigned to $\boldsymbol{A}$. This being the case, it is not surprising that that which is given by the definition of `U` is not provable by the Z2 system, since it is not a definition of a Z2 formula.

Shankar does not give any details regarding the second argument of `EV`, but clearly, if the Z2 formula is precisely definable, and the Lisp expression is precisely definable, there must be a precisely definable function[i] j, such that `(LIST f-p 0 1)` = $j(\boldsymbol{A})$, which gives us the Z2 formula $\boldsymbol{A}$ as:

$$\boldsymbol{A}\begin{cases}\ulcorner \texttt{0} \urcorner \\ \ulcorner j(\boldsymbol{A}) \urcorner \\ \langle\ \ulcorner \texttt{(GCODE A)} \urcorner,\ \ulcorner \texttt{(GCODE GIVEN))} \urcorner\ \rangle \\ \ulcorner \texttt{(FA)} \urcorner \\ \ulcorner \texttt{(VAL)} \urcorner\end{cases}$$

and $\boldsymbol{G}$ as:

$$\boldsymbol{G}\begin{cases}\ulcorner \texttt{0} \urcorner \\ \ulcorner j(\boldsymbol{A}) \urcorner \\ \langle\ y,\ \ulcorner \texttt{(GCODE GIVEN))} \urcorner\ \rangle \\ \ulcorner \texttt{(FA)} \urcorner \\ \ulcorner \texttt{(VAL)} \urcorner\end{cases}$$

and the 'undecidable sentence' as:

$$\boldsymbol{G}\begin{cases}\ulcorner \texttt{0} \urcorner \\ \ulcorner j(\boldsymbol{A}) \urcorner \\ \langle\ \ulcorner \texttt{(GCODE G GIVEN)} \urcorner,\ \ulcorner \texttt{(GCODE GIVEN))} \urcorner\ \rangle \\ \ulcorner \texttt{(FA)} \urcorner \\ \ulcorner \texttt{(VAL)} \urcorner\end{cases}$$

which is not a sentence at all, but a formula with a free variable $\boldsymbol{A}$.

Shankar's fundamental error arises from his assumption that he can use a Lisp expression `(LIST f-p 0 1)` that is a fixed value as a term that represents the concept of not one specific Z2 formula, but a set of Z2 formulas. That Lisp expression should be a variable term in the context of its usage, but it is not, and that error renders Shankar's proof invalid.

[i] Naturally, this function is definable in a meta language to the Lisp language

3 Response from Shankar to the demonstration of the error in his proof

Shankar's response to the demonstration of an error in his proof makes no attempt to rebut the demonstration of an error in his proof. He writes:

> *'Gödel's Theorem is hardly a typical theorem in the broad mathematical landscape, and clearly computer verification of its proof presents an unusual set of issues. ... the issues raised in the article amount to mere nit-picking and indicate a fundamental misunderstanding of Gödel's original argument.'*

To nit-pick is to find an minor inconsequential defect. Shankar provides no logical argument to show how the confusion of a variable and a non-variable might be considered inconsequential. When Shankar claims that this author has a 'fundamental misunderstanding of Gödel's original argument', one might reasonably expect that Shankar would provide some evidence to support this assertion; but Shankar declines to provide any such evidence.

References

[1] N. Shankar, *Metamathematics, Machines, and Gödel's proof.* Cambridge University Press, 1997. ISBN: 9780521585330.

[2] Boyer-Moore, "The Boyer-Moore Theorem Prover (NQTHM)."

[3] ACL2, "The ACL2 software system." `http://www.cs.utexas.edu/users/moore/acl2/`.

Bibliography - Errors in other incompleteness proofs

[a] J. R. Meyer. *"An Error in a Computer Verified Proof of Incompleteness by John Harrison."* `http://www.jamesrmeyer.com/pdfs/ff_harrison.pdf`, 2011

[b] J. R. Meyer. *"An Error in a Computer Verified Proof of Incompleteness by Russell O'Connor."* `http://www.jamesrmeyer.com/pdfs/ff_oconnor.pdf`, 2011

[c] J. R. Meyer. *"A Fundamental Flaw in an Incompleteness Proof in the book 'An Introduction to Gödel's Theorems' by Peter Smith."* `http://www.jamesrmeyer.com/pdfs/ff_smith.pdf`, 2011

[d] J. R. Meyer. *"A Fundamental Flaw In Incompleteness Proofs by Gregory Chaitin."* `http://www.jamesrmeyer.com/pdfs/ff_chaitin.pdf`, 2011

[e] J. R. Meyer. *"A Fundamental Flaw In Incompleteness Proofs by S. C. Kleene."* `http://www.jamesrmeyer.com/pdfs/ff_kleene.pdf`, 2011