AN ERROR IN A COMPUTER VERIFIED PROOF OF INCOMPLETENESS BY RUSSELL O'CONNOR

James R Meyer
http://www.jamesrmeyer.com

v1 03 November 2011

Abstract

This paper examines a proof of incompleteness by Russell O'Connor, 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 Russell O'Connor, first published in 2005 [1]. This proof also can be found in a thesis by O'Connor [2]. 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 O'Connor's proof

O'Connor's incompleteness proof uses the Coq proof assistant [3], a system that has gained much acceptance. O'Connor defines a formal system within Coq, which is a system similar to Peano arithmetic. It consists of a set of symbols which constitute a language that O'Connor calls LNN, and a set of axioms which he calls the NN axioms. O'Connor's description of his proof can be found online [1].

It should be noted that O'Connor claims that his formal proof is a proof of incompleteness for any formal system satisfying certain criteria. This is factually incorrect. O'Connor's formal proof is a proof of incompleteness only of one particular formal system, the above mentioned system with the NN axioms; his formal proof does not extend the proof to apply to any other formal system.

For clarity, O'Connor's computer code expressions will be printed here in monospace typewriter font, for example:

```
notH (forallH 0 (forallH 1 (equal (var 0) (var 1))))
```

represents the formula $\neg\forall x_0.\forall x_1.x_0 = x_1$.

We shall first summarize some of the notation and definitions that O'Connor uses before dealing with the details of his proof:

$\ulcorner\ \urcorner$ Confusingly, O'Connor uses this notation to represent two different concepts:

- One meaning is that $\ulcorner n \urcorner$ represents a function that gives the formal system format of a given natural number n.[a] O'Connor refers to this as the function `natToTerm : nat -> Term`, so that, for example, $\ulcorner 0 \urcorner = \mathbf{0}$, $\ulcorner 1 \urcorner = \mathbf{S0}$, etc.
- The other meaning is that $\ulcorner \phi \urcorner$ represents the Gödel numbering function of a symbol combination ϕ of the formal system.[b] In this case $\ulcorner \phi \urcorner$ is in fact the function $\ulcorner$`codeFormula` $\phi \urcorner$ where `codeFormula` is a function that gives a unique number for every formal formula ϕ, and similarly $\ulcorner t \urcorner$ is really the function $\ulcorner$`codeTerm` $t \urcorner$ where `codeTerm` is a function that gives a unique number for every term t in the formal system. In this paper, for clarity we shall use $\ulcorner\ \urcorner$ only to refer to the function `natToTerm`, so that the value of $\ulcorner n \urcorner$ is the formal system format of the natural number n.

$\phi[x_i/s]$ This represents the substitution of the variable x_i in the formula ϕ by the value s.[c]

In his proof, O'Connor states:

> *'I proved that substitution is primitive recursive. Since substitution is defined in terms of Formula and Term, it itself cannot be primitive recursive. Instead I proved that the corresponding function operating on codes is primitive recursive. This function is called* `codeSubFormula` *and I proved it is correct in the following sense:*
>
> `codeSubFormula(`$\ulcorner$`codeFormula`$\phi \urcorner, i, \ulcorner$`codeTerm`$s \urcorner$`) =` $\ulcorner \phi[x_i/s] \urcorner$ '

O'Connor's proof relies on the theorem that for every primitive recursive number-theoretic relation or function there is a corresponding representation of that relation or function in the formal system. A crucial step in O'Connor's proof is the assertion that this applies for the function: `codeSubFormula(`$\ulcorner$`codeFormula`$\phi \urcorner, i, \ulcorner$`codeTerm`$s \urcorner$`)`.

Now, according to the definition of a primitive recursive number-theoretic function[d], for any expression that is a primitive recursive number-theoretic function, all of its terms must also be primitive recursive number-theoretic functions, or be a natural number, or be a variable with the domain of natural numbers. For example, given that the function $f(y) = ((y-7)-5)-3$ is primitive recursive then so also is the term $y-7$. The function $f(y) = (y-7)$ is primitive recursive, and the function $f(z) = (z-5)-3$, derived by letting $z = y-7$, is also primitive recursive. Similarly, for the term $z-r$, $f(z) = z-r$ is primitive recursive as is the function $f(x) = x-3$, derived by letting $x = z-5$.

The error in O'Connor's proof is readily apparent. The function that is `codeSubFormula(`$\ulcorner$`codeFormula`$\phi \urcorner, i, \ulcorner$`codeTerm`$s \urcorner$`)` cannot be a number-theoretic function, since the terms $\ulcorner$`codeFormula`$\phi \urcorner$ and $\ulcorner$`codeTerm`$s \urcorner$ are not number-theoretic terms.

[a] See O'Connor's section 2.8 *Languages and Theories of Number Theory*

[b] See O'Connor's section 3 *Coding*

[c] See O'Connor's section 2.4 *Definition of substituteFormula*

[d] As a point of interest, a definition of primitive recursion is found in Gödel's proof of incompleteness [4].

The domain of the variable ϕ is not the domain of natural numbers, and therefore the term $\ulcorner$`codeFormula`$\phi\urcorner$ does not satisfy the definition of a primitive recursive number-theoretic function. The term $\ulcorner$`codeFormula`$\phi\urcorner$ is a substitution of a variable of the function `codeSubFormula`. It follows that either the domain of that variable is not restricted to terms which are natural numbers or number-theoretic terms and that `codeSubFormula` cannot be a primitive recursive number-theoretic function, or else the substitution of the variable of the function `codeSubFormula` was an illegitimate operation. The same applies for the term $\ulcorner$`codeTerm`$s\urcorner$, which is also a substitution for a variable of the function `codeSubFormula`.

Since the function `codeSubFormula` cannot be both a primitive recursive number-theoretic function and also have variables with a domain that includes the entities $\ulcorner$`codeFormula`$\phi\urcorner$ and $\ulcorner$`codeTerm`$s\urcorner$, O'Connor's proof is critically flawed.

O'Connor refers to extensionally equivalent functions. He states:

> *'Rather than working directly with primitive recursive expressions, I worked with particular Coq functions and proved they were extensionally equivalent to the evaluation of primitive recursive expressions.'*

Of course, there can be a function called `codeSubFormula`(x, y, z) that is not a number-theoretic function, and where the variables x, y, and z may be substituted by $x = \ulcorner$`codeFormula`$\phi\urcorner$, $y = i$, and $z = \ulcorner$`codeTerm`$s\urcorner$ (for allowable values of ϕ and s), and that gives the expression: `codeSubFormula`$(\ulcorner$`codeFormula`$\phi\urcorner, i, \ulcorner$`codeTerm`$s\urcorner)$.

Unsurprisingly, for the natural number values a, b and c, as given by $a = \ulcorner$`codeFormula`$\phi\urcorner$, $b = i$, and $c = \ulcorner$`codeTerm`$s\urcorner$, there will be a number-theoretic function $F(u,v,w)$ (where the domain of the variables u, v, w are natural numbers or number-theoretic functions) so that the function $F(a,b,c)$ gives the same natural number value as the function `codeSubFormula`$(\ulcorner$`codeFormula`$\phi\urcorner, i, \ulcorner$`codeTerm`$s\urcorner)$.

And one can apply the name `codeSubFormula`(x, y, z) to the function $F(u,v,w)$. But that does not mean that the two functions which now have the same name are extensionally equivalent, nor does it mean that the two functions must both be primitive recursive number-theoretic functions. Two functions are extensionally equivalent if and only if they *always* give the same value for the same argument values. This obviously cannot be the case for the function `codeSubFormula`$(\ulcorner$`codeFormula`$\phi\urcorner, i, \ulcorner$`codeTerm`$s\urcorner)$ which is not a number-theoretic function, and $F(u,v,w)$, which is a number-theoretic function. The fact is that the first and third arguments of these two functions do not have the same domains, and the function $F(u,v,w)$ is undefined for terms outside the domain of its variables.

3 Response from O'Connor to the demonstration of the error in his proof

O'Connor's response to the demonstration of an error in his proof is as follows:

> *'Essentially the author's claim is that O'Connor proves that* `codeSubFormula(`$\ulcorner$`codeFormula`$\phi\urcorner, i, \ulcorner$`codeTerm`$s\urcorner$`)` *is a primitive recursive function. Of course, O'Connor doesn't claim this in O'Connor's paper nor in his formal proof. As the author rightly notes, this expression as a function of phi and doesn't even have the type of a number theoretic function. O'Connor does claim and formally prove that* `codeSubFormula` *is a primitive recursive function.*
>
> *The author is under the mistaken impression that one needs to prove that* `codeSubFormula(`$\ulcorner$`codeFormula`$\phi\urcorner, i, \ulcorner$`codeTerm`$s\urcorner$`)` *is primitive recursive in order to complete Gödel's theorem. However, to prove Gödel's incompleteness theorem, it suffices to prove* `codeSubFormula` *is primitive recursive and to prove:*
>
> **Lemma** `codeSubFormulaCorrect :`
> `forall (f : Formula) (v : nat) (s : Term),`
> `codeSubFormula (codeFormula f) v (codeTerm s) =`
> `codeFormula (substituteFormula L f v s).'`
> *which is exactly what O'Connor formally proves, and is what Gödel intended.'*

Here, O'Connor ignores elementary principles of mathematics. If the fundamental principles of mathematics are being rigorously adhered to, then the free variable of a function cannot be substituted by values that lie outside the domain of that variable. If O'Connor's result depends on violating the fundamental principles of mathematics, then it cannot be described as a mathematical proof.

If a function is defined as being a number-theoretic function, then it is defined as having variables with a clearly defined domain, a domain of natural numbers or functions that are themselves number-theoretic functions. If a function that has the designation `codeSubFormula` is a primitive recursive number-theoretic function, then it cannot be the same function that happens to have the same designation, but whose variables have been substituted by terms that are not natural numbers or number-theoretic functions. The fact is that the function `codeSubFormula` that O'Connor uses in his proof is a function where some of its variables have a domain that is not restricted to the domain of natural numbers or number-theoretic terms, and therefore it ***must*** be the case that that function is not a primitive recursive number-theoretic function. Logically, his proof cannot contain a function that is called `codeSubFormula` which is a primitive recursive number-theoretic function, and at the same time, the variables of that function are substitutable by non-number-theoretic terms.

O'Connor continues:

> *'In any case, if any serious researcher claimed that a software proof assistant is flawed, they would illustrate the flaw in the software system itself, not attack specific theorems proved using such software.'*

This author has demonstrated that there is a flaw in the proof, and hence there must be a flaw in the verification of the proof. The system software consists of thousands of lines of computer code, and the proof itself consists of thousands of lines of computer code. The complete definition of any term can be dependent on many other lines of code. For example, the code for the function `codeSubFormula` depends on the following sub-sections of code: `primRec`, `cPair`, `Arith`, `folProp`, `code`, `extEqualNat`, `Bvector`, `codeSubTerm`, `codeFreeVar`, `Max` and many of these depend on other sub-sections. For example, `primRec` is dependent on: `Arith`, `Peano_dec`, `Compare_dec`, `Coq.Lists.List`, `Eqdep_dec`, `extEqualNat`, `Bvector`, `misc`, `Even`, `Max`. It is rather bizarre to suggest that it is incumbent upon this author to trawl through that material to locate the precise lines of codes where the error originates, rather than the creators of the system software or the proof.

References

[1] R. O'Connor, "*Essential Incompleteness of Arithmetic Verified by Coq.*" `http://arxiv.org/abs/cs/0505034`, 2005.

[2] R. O'Connor, "*Incompleteness & Completeness.*" `http://webdoc.ubn.ru.nl/mono/o/oconnor_r/incoanco.pdf`, 2009.

[3] Coq, "The Coq Proof Assistant." `http://coq.inria.fr/refman/index.html`.

[4] K. Gödel, "Über formal unentsceidebare sätze der Principia Mathematica und verwandter Systeme I," *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173–198, 1931.

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 Natarajan Shankar."* `http://www.jamesrmeyer.com/pdfs/ff_shankar.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