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

James R Meyer

http://www.jamesrmeyer.com

v2: 03 June 2016

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.

Version History:

Version 2: This version deals with anomalies arising from O'Connor's usage of the notation ⌜ ⌝ in his descriptive text, which is rather confused. This does not affect the thrust of the argument presented. An addendum has been added which explains how such a flaw can occur in a proof which is claimed to have been completely checked by computer code. The previous version can be accessed via *http://www.jamesrmeyer.com/sitemap.html*.

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:

$\ulcorner\ \urcorner$ O'Connor's use of this notation is rather confused. It only appears in his description of the computer proof, and not in the computer code, so it is intended to refer to functions in the computer code.

In one section he states that he will use $\ulcorner n \urcorner$ in the text to refer to the function `natToTerm` in the computer code, which is a function that gives the formal system format of a given natural number n, so that, for example, $\ulcorner 0 \urcorner$ = **0**, $\ulcorner 1 \urcorner$ = **S0**, etc.[a]

In another section he defines that he will use $\ulcorner \phi \urcorner$ in the text to represent the function $\ulcorner$`codeFormula` $\phi \urcorner$, and similarly $\ulcorner t \urcorner$ is to represent the function $\ulcorner$`codeTerm` $t \urcorner$.[b]

However, it cannot be that $\ulcorner$`codeFormula` $\phi \urcorner$ refers to a composite function where the function `natToTerm` is applied to the value of `codeFormula` ϕ, since such usage does not occur in the actual computer code. The same applies to $\ulcorner$`codeTerm` $t \urcorner$ and `codeTerm` t.

It is evident that O'Connor's intention is that where $\ulcorner\ \urcorner$ occurs in the text, it corresponds to the computer code for the Gödel numbering function; where $\ulcorner \phi \urcorner$ occurs in the text, and ϕ is a formal system formula, then it corresponds to the computer code `codeFormula` ϕ, which is the Gödel numbering function for a formula, and where $\ulcorner t \urcorner$ occurs in the text, and t is a term of the formal system, it corresponds to the computer code `codeTerm` t, which is the Gödel numbering function for a term.[c]

$\phi[x_i/s]$ This represents the substitution of the variable x_i in the formula ϕ bythe value s.[d] As indicated above, where $\ulcorner \phi[x_i/s] \urcorner$ appears in the text , this is intended to represent the (Gödel) number of the formula tht results from such substitution.

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(`$\ulcorner \phi \urcorner, i, \ulcorner s \urcorner$`)` $= \ulcorner \phi[x_i/s] \urcorner$ '

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

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

[c] For convenience, the term 'Gödel numbering' is used here to refer to the encoding function that assigns a unique number to each symbol string of the formal system, although O'Connor uses a different encoding system to that which Gödel used.

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

Assigning the computer code names for the functions on the left side of the above gives:
codeSubFormula(codeFormula ϕ, i, codeTerm s) = $\ulcorner \phi[x_i/s] \urcorner$

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(codeFormula ϕ, i, codeTerm s).

Now, according to the definition of a primitive recursive number-theoretic function[e], 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 composite function that is codeSubFormula(codeFormula ϕ, i, codeTerm s) cannot be a number-theoretic function, since the terms codeFormula ϕ and codeTerm s are not number-theoretic terms.

The domain of the variable ϕ is not the domain of natural numbers, and therefore the term codeFormula ϕ does not satisfy the definition of a primitive recursive number-theoretic function. The term codeFormula ϕ has been substituted for 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, in which case 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 codeTerm s, which has also been substituted 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 codeFormula ϕ and codeTerm s, 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 = codeFormula ϕ, $y = i$, and z = codeTerm s (for allowable values of ϕ and s), and that gives the composite function:
codeSubFormula(codeFormula ϕ, i, codeTerm s).

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

Unsurprisingly, for specific natural number values a, b and c, as given by a = `codeFormula` ϕ, $b = i$, and c = `codeTerm` s, 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 composite function `codeSubFormula(codeFormula` ϕ, i, `codeTerm` s`)`.

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 composite function `codeSubFormula(codeFormula` ϕ, i, `codeTerm` s`)` 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(codeFormula` ϕ, i, `codeTerm` s`)` *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(codeFormula` ϕ, i, `codeTerm` s`)` *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 unentsceidbareś ätze 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

ADDENDUM
to
"An Error in a Computer Verified Proof of Incompleteness by Russell O'Connor"

03 June 2016

This addendum was created because it had become apparent that some readers had erroneous impressions regarding O'Connor's proof, leading them to the erroneous conclusion that the entirety of O'Connor's proof is a complete self-contained formal proof of incompleteness based only on fundamental mathematical axioms. This is not actually the case, as will be demonstrated below.

In addition, this addendum also addresses O'Connor's contention that: *'... 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.'*

Addendum: 1 Variable Domains and Substitutions

In many incompleteness proofs, there is an assumption that one can always substitute a variable that has a domain of natural numbers by a non-number-theoretic function provided its range of values are natural numbers. As will be considered below, if it is the case that all variables of the new expression should also have the domain of natural numbers, this is logically inadmissible. The question that is addressed here is whether such substitution can give rise to errors, and it will be shown that errors can arise when it is assumed that the new expression produced by the substitution retains the number-theoretic property of the original expression.

Consider this expression that is defined to be purely number-theoretic:

$$n+2>0$$

Now, given a function $GN(s)$, whose range is natural numbers, and where s is a variable whose domain is a given set of symbol strings that include strings that are not numbers,[a] suppose that we now substitute the n by $GN(s)$ to give the expression:

$$GN(s)+2>0$$

Such a substitution is a common mathematical operation, but this is merely a convenient short-cut to a more formal expression:

> *For all s as symbol strings, there exists a natural number n, such that* $n = GN(s)$*, and* $n+2>0$.

The crucial point here is that this is an expression where its ***overall*** language is not confined to objects that are only natural numbers, that is, it is not a purely number-theoretic expression, but within this overall expression, $n+2>0$ remains as a purely number-theoretic expression. It follows that the convenient short-cut that is:

$$GN(s)+2>0$$

is an expression whose overall language is not confined to objects that are only natural numbers, and it is not a number-theoretic expression. It is an expression where the variable n,

a. The Gödel numbering function is an example of such a function

whose domain is natural numbers, has been substituted by an expression that is not a natural number. Clearly if $n+2>0$ has a property that is *dependent* on the fact that it is a purely number-theoretic expression, then there is no reason to assume that that property also applies to $GN(s)+2>0$.

As noted, such a substitution is an extremely common mathematical operation. In cases that involve only numbers and variables for natural numbers, the function that is being substituted is a function which itself is a number-theoretic function whose objects are natural numbers, and whose variables have the domain of natural numbers. As an example, if we use the same purely number-theoretic expression as was used above:

$n+2>0$

and given a purely number-theoretic function $f(x)$, where x is a variable whose domain is natural numbers, and we now substitute the n by $f(x)$ to give the expression:

$f(x)+2>0$

Again this is a convenient short-cut to a more formal expression:

For all x, there exists n, such that $n=f(x)$, *and* $n+2>0$.

But here, in contrast to the previous case, the overall language of the expression remains as a purely number-theoretic language. In this case, we do not need to separately specify the domains of n and s since every variable in the language is already specified as having the domain of natural numbers.

Addendum: 2 Representability

O'Connor's proof, in common with many other incompleteness proofs, relies on the notion of "representability" of certain number-theoretic expressions that are not expressions of the formal language NN[b] - that is, that for certain number-theoretic expressions of the computer code[c] there are expressions of the formal system NN that "represent" the relations between numbers that are given by those expressions of the computer code. More formally, the notion of "representability" is generally given in terms such as the following:

An n-ary number-theoretic relation $R(x_1,x_2,...,x_n)$ is said to be representable in a formal number-theoretic system T if there is an expression with n free variables in the system T, which we designate by $\varphi(m_1,m_2,...,m_n)$, and where:

$\forall x_1,x_2,...,x_n,\ \exists m_1,m_2,...,m_n,$

$m_1=F_T(x_1),\ m_2=F_T(x_2),...,\ m_n=F_T(x_n)\ \wedge$

$R(x_1,x_2,...,x_n)\Rightarrow T\vdash\ \varphi(m_1,m_2,...,m_n)\ \wedge\ \neg R(x_1,x_2,...,x_n)\Rightarrow T\vdash\ \neg\varphi(m_1,m_2,...,m_n)$

where F_T is a function that gives the numbers $x_1,x_2,...,x_n$ in the format of the formal system T.

Similarly, an n-ary number-theoretic function $f(x_1,x_2,...,x_n)$ is said to be representable in a formal number-theoretic system T if there is an $n+1$-ary number-theoretic relation $R(x_1,x_2,...,x_{n+1})$ where:

$\forall x_1,x_2,...,x_{n+1},\ \ R(x_1,x_2,...,x_{n+1})\ \equiv\ x_{n+1}=f(x_1,x_2,...,x_n)$

and the above condition on $R(x_1,x_2,...,x_{n+1})$ also applies.

b. NN is the formal system for which O'Connor's proof claims incompleteness.

c. i.e. expressions that are not expressions of the formal language NN.

Note that in the above, the only variables involved are variables whose domain is natural numbers. The notion of "representability" will be further considered in sections Addendum: 2.1 and Addendum: 3 below. It will be noted here that the methodology of O'Connor's proof is the same as that used in several other incompleteness proofs, for example Smith[1], Smullyan[2],[3], Boolos[4], Franzén[5], Lindström[6], Mostowski[7], Harrison[8]. In such proofs, the principle is that, given an expression that is a number-theoretic expression that states a relation between numbers and/or variables whose domain is natural numbers, then (subject to certain additional conditions) there is an expression in the formal system that "represents" the same relation.

Addendum: 2.1 Number-theoretic expressions

One of the conditions that is the basis for the notion that if a relation is "representable", and so there is an expression in the formal system NN that states that relation, is that the relation that is "representable" and the formal system in which the "representation" is to occur are both number-theoretic. A purely number-theoretic system is a system where the only objects are natural numbers, and all variables have a domain that consists of the objects of the system, (i.e., natural numbers in the format of the system). Given an expression of the system:

1. natural numbers may be substituted for a variable in an expression of the system.

And depending on the rules of the particular system, it is also the case that:

2. a valid expression of the system can also be produced by replacing a variable x in an existing expression of the system by a different variable y of the system, providing certain conditions are observed (such as the new variable name not conflicting with variables already in the expression). This is basically a variable name change.
3. a valid expression of the system can also be produced by replacing a variable x in an existing expression A of the system by another existing number-theoretic expression B of the system that is a functional expression[d] providing certain conditions are observed (such as there may not be variables in the expression B that conflict with variables already in the expression A).

Addendum: 2.2 Types in O'Connor's computer code

O'Connor's proof uses types, one of the types being "nat", and which is *similar but not identical* to the notion of natural number. In O'Connor's code, variables that are defined as type "nat" can be substituted by natural numbers, which are also assigned the type "nat". But, crucially, such variables can also be substituted by functions that have the *range* of natural numbers, irrespective of whatever objects are referenced by the function, and irrespective of the domains of the variables of the function.

It follows that the type "nat" does not correspond precisely to the definition of natural numbers as they occur within the formal system NN. In the formal system NN, a variable cannot be substituted by a function which references objects that are not natural numbers; it cannot be substituted by a function whose variables do not have the domain of natural numbers. Any such substitution would result in an expression that is not an expression of the formal system NN.

d. An expression that has free variables that have the domain of natural numbers, and when those variables are substituted by specific natural numbers, the resultant expression evaluates as a specific natural number.

Addendum: 3 Assumptions in O'Connor's proof

O'Connor does not actually provide a proof of "representability" from fundamental mathematical principles; instead he gives definitions of "representability" which include assumptions about what expressions are "representable", and then he proves that certain terms of his computer code satisfy those definitions. Hence the result of his proof depends on the assumptions that are included in those definitions.

When O'Connor claims that his proof is a fully computer checked proof, one expects that it should not include any non-axiomatic assumptions. One could, of course, simply reject O'Connor's "computer proof" as not being a completely proved proof, since it is not a complete proof from fundamental mathematical principles alone; the assumptions that are implicit in O'Connor's definition of "representability" are not commonly accepted axiomatic assumptions. Nevertheless, one might accept the result of O'Connor's proof - but only if one is completely satisfied that such assumptions are as acceptable as a standard fundamental mathematical axiom. However, as will be shown below, a logical analysis of O'Connor's proof demonstrates that his implicit assumptions are neither logical, reasonable, nor acceptable in a rigorous mathematical proof.

For a definition of "representability" to be logically reasonable and acceptable, one would expect that any expression of a relation/function for which there is a claim that there must also be an expression in the formal system NN that states that relation/function, then it should also conform to the three rules above.[e] Otherwise if, for example, we construct an expression L by replacing a variable x in an existing number-theoretic expression K by an expression which includes a variable whose domain is not natural numbers, then that expression L does not conform to Condition 3 above, and thus there can be no logical reason to suppose that the derived expression L is "representable" in the formal system. In other words, we would have no reason to accept an assumption that such a derived expression as L is "representable" in the formal system, and we would reject the notion that such an assumption would be a valid assumption in a mathematical proof.

The suggestion might be made that this author has not proved that there cannot be expressions that do not conform to the three conditions above, but which there might nevertheless be such that they satisfy the definition of "representability" and that there might also be an equivalent expression in the formal system NN. This is correct, but the onus, as in all proofs, is on the creator of a proof that relies on his definition of "representable" to actually prove with complete rigor that there is also an equivalent expression in the formal system NN. A reliance on definitions that include implicit assumptions can only be acceptable if the assumptions can be shown to be logically reasonable.

Addendum: 3.1 O'Connor's implicit assumptions in "representability"

There is a substantial problem in attempting to consider from O'Connor's computer code definitions of "representability" whether the assumption that if an expression satisfies the computer code definitions of "representability" then there necessarily exists an equivalent expression in the formal system NN is a logically reasonable and acceptable assumption. This is because in the computer code "representability":

e. Depending on the actual formal system, there may be other conditions that must also be observed; the above is the minimum requirement that applies to all number-theoretic formal systems.

1. is defined in terms of several code expressions that are themselves recursively defined in several levels of computer code

and

2. one needs to know precisely how the definitions interact with the Coq computer code system; that is, one would need to have an in-depth knowledge of the Coq system.

Two of the primary definitions involved in O'Connor's definition of "representability" are:

```
Definition Representable (n : nat) (f : naryFunc n)
    (A : Formula) : Prop :=
    (forall v : nat, In v (freeVarFormula LNN A) -> v <= n) /\
    RepresentableHelp n f A.
```

and

```
Definition RepresentsInSelf (T:System) :=
  exists rep : Formula, exists v : nat,
  (forall x : nat, In x (freeVarFormula LNN rep) -> x = v) /\
  (forall f : Formula,
    mem Formula T f ->
    SysPrf T (substituteFormula LNN rep v (natToTerm (codeFormula f)))) /\
  (forall f : Formula,
     mem Formula T f ->
    SysPrf T
      (notH (substituteFormula LNN rep v (natToTerm (codeFormula f))))).
```

and examples of other terms that are involved in the definition of "representability" include:

```
RepresentableHelp, RepresentablesHelp, RepresentableAlternate,
RepresentableHalf1, RepresentableHalf2, ExpressibleHelp, RepresentablesHelp,
Representable_ext.
```

So, on the face of it, it is exceedingly difficult for most persons other than those who have in-depth knowledge of the Cog computer system to reach an informed positive decision that the assumption is acceptable. However, by approaching the problem from a different perspective, we can reach a well-informed decision that the assumption is certainly not acceptable. If we can observe that O'Connor's usage of "representability" does not actually reflect any acceptable logical principles regarding what computer code expressions can be precisely "represented" in the formal system NN, then there is no logical reason why one should consider that the assumption is acceptable.

In particular, there is no logical reason why one should accept O'Connor's lemma:

```
Lemma codeSubFormulaCorrect:
    forall (f : Formula) (v : nat) (s : Term),
         codeSubFormula(codeFormula f) v (codeTerm s) =
         codeFormula(substituteFormula L f v s).
```

O'Connor has proved that the formula `codeSubFormula(u,v,w)` satisfies his definition of "representability". He claims that this proves that it is primitive recursive; therefore it must be purely number-theoretic. Now, if the expression `codeSubFormula(u,v,w)` is actually "representable", then there is a formula of the formal system that satisfies that definition. If that is the case, then the inverse of such "representability" must likewise apply to that formula of the formal system. From that it follows that any computer code expression for which the inverse of "representability" applies must state only the relation that is stated by the formal system, and not invoke any additional

properties over and above those that are inherent in the formula of the formal system. In particular, in any such computer code expression, the variables of that expression must conform to the three conditions noted above.

But O'Connor claims that because the function `codeSubFormula(u,v,w)` satisfies his definition of "representability", then there must be an expression in the formal NN that "represents" the function:

```
codeSubFormula(codeFormula f) v (codeTerm s).
```

And, furthermore, he claims that because the composite function:

```
codeSubFormula(codeFormula f) v (codeTerm s)
```

is extensionally equivalent to the composite function:

```
codeFormula(substituteFormula L f v s)
```

that there must be an expression in the formal NN that "represents" the composite function

```
codeFormula(substituteFormula L f v s).
```

This is, of course, nonsensical, since in order to "represent" such a function, the formal system NN would have to have a variable that has the same domain as `f` (domain of `f` is formula of a formal system), a variable that has the same domain as `s` (domain of `s` is terms of the formal system), and a variable that has the same domain as `L` (domain of `L` is formal systems), which is impossible.

The implicit assumption in O'Connor's computer code definitions of "representability", and the subsequent usage of those definitions do not reflect any logical principles regarding what can actually be stated in the formal system NN. In O'Connor's proof the computer code allows a variable of type "nat" (such as `u`, `v` and `w` in `codeSubFormula`) to be substituted by an expression such as `codeFormula f`. This expression `codeFormula f` is itself not actually a natural number, that is, it is not an entity that satisfies the Peano axioms, since there is no successor for the expression `codeFormula f`; nor is it a variable whose domain is natural numbers, nor is it itself a number-theoretic expression. Hence it does not satisfy the three conditions above, and it follows that there is no logical reason to suppose that there is an expression in the formal system NN that states the function `codeSubFormula(codeFormula f) v (codeTerm s)`.

Of course, O'Connor's substitution as in his lemma `Lemma codeSubFormulaCorrect` is simply a short-cut to a more formal expression, as referred to in section Addendum: 1 above:

For all `f` *as a formula in NN, for all* `v` *as a natural number, for all* `s` *as a term in NN,*
there exist `u`, `w` *as natural numbers, such that*
`u = codeFormula f, w = codeTerm s`, *and*
`codeSubFormula(u,v,w) = codeFormula(substituteFormula L f v s)`

As noted in section Addendum: 1 this is an expression where its ***overall*** language is not confined to objects that are only natural numbers, but within this overall expression, if it is the case that the function `codeSubFormula(u,v,w)` is a primitive recursive number-theoretic function (as O'Connor claims it is), then `codeSubFormula(u,v,w)` remains within the overall expression as a purely number-theoretic function.

Furthermore, even when a valid expression[f] is substituted for the variable `f` in the expression `codeFormula f`, this still does not mean that the resultant expression is "representable" in the formal system NN. Simply because it happens to be the case that when a valid expression such as `abc` is substituted for the variable `f`, the expression `codeFormula abc` then ***evaluates*** in the computer code as a natural number, that provides no logical reason to suppose that the

f. A formula of the formal system NN is a valid value of the domain of the variable `f`.

substituted expression `codeFormula abc` is itself actually "representable" in the formal system NN. The expression `codeFormula` necessarily requires bound variables whose domain is not natural numbers, but whose domain is the set of all symbols of the formal system. That is, the actual ***evaluation*** of the function for a specific value of its free variable is ***evaluated*** in a system that is ***necessarily*** not a number-theoretic system, so there is no logical reason to suppose that the expression itself is "representable" in the formal system NN - and neither is there any reason to suppose that there can be an ***evaluation*** of the expression in the formal system NN.

Of course, if an author wishes to simply assume that expressions such as the above that do not conform to the three conditions are "representable" in NN, then he can base a "proof" around that assumption, but to claim that the result is a valid proof of incompleteness is an absurdity, and flies in the face of all notions of a rigorous mathematical proof.

Addendum: 4 A remark on computer checked proofs

O'Connor's response to this author's original paper included the contention that there was an onus on this author to illustrate the flaw in the software system itself. It is clear that the error in the proof lies with O'Connor's unacceptable assumptions.

The matter discussed here of an assumption that is obfuscated in the computer code raises an important point in general regarding computer proofs. O'Connor proclaims his proof as a computer checked proof, and it can be said that it has gained considerable acceptance as such,[g] where the general impression seems to be that the entire proof is a complete self-contained formal proof that has been checked by computer. However, it contains assumptions that are not computer checked, and which are not fundamental mathematical axioms.

In this case, the computer checks that the proof proves something, which it does. However, it is quite clear that what it actually proves and what O'Connor claims that it proves are two quite different propositions. Because of the implicit assumption in O'Connor's definitions of "representability", O'Connor's "representability" does not precisely capture the conditions that must be observed for there is to be a reasonable claim that a non-formal system expression is "representable" in the formal system.

This makes a mockery of the entire raison d'être of computer checked proofs, which is that they should eliminate all errors that arise from informal reasoning. If the author of a computer proof can simply slip in an assumption into a definition, therein lies a back-door by which erroneous informal reasoning can be introduced. Such back-doors should be totally eliminated if one is to be able to have any confidence in any claim of a computer proof.

Until such back-doors are eliminated, any assumptions such as O'Connor has included should surely be clearly and explicitly stated and included in any reference to the proof being computer checked. If O'Connor's proof has been commonly accepted as a fully formal proof that has been computer checked, one is given to wonder how many such hidden assumptions are included in other claims of computer checked proofs.

g. O'Connor's paper describing his proof has been accepted and published by Springer, and has been cited in several journal articles.

Addendum References

[1] P. Smith, *An Introduction to Gödel's Theorems*. Cambridge University Press, 2006. ISBN: 9780521857840.

[2] R. M. Smullyan, *Gödel's Incompleteness Theorems*. Oxford University Press, 1992. ISBN: 0195046722.

[3] R. M. Smullyan, *Recursion Theory for Metamathematics*. Oxford University Press, 1993. ISBN: 9780195082326.

[4] G. Boolos, J. Burges, and R. Jeffrey, *Computability and Logic*. Cambridge University Press, fifth ed., 2007. ISBN: 9780521877527.

[5] T. Franzén, *Gödel's Theorem: An Incomplete Guide to its Use and Abuse*. A K Peters, 2005. ISBN: 1568812388.

[6] P. Lindstrom, *Lecture Notes: Aspects of Incompleteness*. Springer-Verlag, 1997. ISBN: 3540632131.

[7] A. Mostowski, *Sentences Undecidable in Formalized Arithmetic*. Greenwood Press, 1982. ISBN: 9780313231513.

[8] J. Harrison, *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009. ISBN: 9780521899574 (eBook format: ISBN: 9780511508653).