Suppose that $\text{DoesHalt( Program, Input )}$ is a program that does its best to determine if Program halts on Input. It has the following property: if it returns true then Program halts on Input, if it returns false then Program does not halt on input. There may be some cases when it returns nothing, but whenever it is humanly possible to make a determination it will make it. And when it does make a determination it is always right!

Now let us construct a program

$$\text{HaltsOnItself( Program )} \{$$
$$\text{DoesHalt( Program, Program )}$$
$$\}$$

It tells us whether Program halts on itself. Let us further construct program

$$\text{AntiDiag( Program )} \{$$
$$\text{if ( HaltsOnItself( Program ) )}$$
$$\text{loop}$$
$$\text{else}$$
$$\text{halt}$$
$$\}$$

What happens if we pass AntiDiag to AntiDiag()?

$$\text{AntiDiag( AntiDiag )} \{$$
$$\text{if ( HaltsOnItself( AntiDiag ) )}$$
$$\text{loop}$$
$$\text{else}$$
$$\text{halt}$$
$$\}$$

What is $\text{HaltsOnItself( AntiDiag )}$ going to do? If it answers either true or false it will contradict itself and lose its status of infallibility. It will wisely stay silent. Most likely it will go into infinite recursion.

When inspecting $\text{AntiDiag( AntiDiag )}$ we notice that we are passing AntiDiag as a
parameter to itself. Can we economize? In fact we can.

\[
\text{AntiSelf(void) } \{
    \text{AntiDiag( AntiDiag )}
\}
\]

The above is a self-referential self-contradictory program. Now suppose that

\[
\text{DoesHalt( HaltsOnItself, AntiSelf )}
\]

halts and says no. Is this possible?

Since \text{HaltsOnItself( AntiSelf )} does not halt it does not say anything in particular about \text{AntiSelf().} But the recursion theorem seems to say that if \text{HaltsOnItself}(\text{AntiSelf}) does not halt then \text{AntiSelf()} must not halt. Actually the recursion theorem says that if \text{HaltsOnItself( AntiSelf) } halts with a certain answer then \text{AntiSelf()} must halt with the same answer and vice versa. It does not say that if there is a proof that \text{HaltsOnItself( AntiSelf) } does not halt there must be a proof that \text{AntiSelf()} does not halt.

All the programs we have been discussing are listed in the table below. The programs in each column are equivalent to each other.
Let us put the above in a perspective. Gödel's sentence is self-referential self-contradictory just as \textbf{AntiSelf()}. The table below shows the comparisons.

| Computability            | Arithmetic                                      |
|--------------------------|-------------------------------------------------|
| \textbf{DoesHalt( HaltsOnItself, AntiSelf )} | \textbf{~(Ex)Prf(x, <#~(Ex)(Prf(x, <#~(Ex)(Ey)( (Prf(x,y) & This(y) )#> )#> )#> )#> )} |
| \textbf{HaltsOnItself( AntiSelf )}          | \textbf{~(Ex)(Prf(x, <#~(Ex)(Ey)( (Prf(x,y) & This(y) )#> )#> )} |
| \textbf{AntiSelf()}             | \textbf{~(Ex)(Ey)( (Prf(x,y) & This(y) )} |

Table 2

[It should be clear that 'Prf(x,y)' means that x is is (the Gödel number of) a proof of (the Gödel number of) y. '<#P#>' means the Gödel number of P. The reason for the unorthodox notation is to make the multiple nesting more readable. 'This(y)' is satisfied only by <#~(Ex)(Ey)( (Prf(x,y) & This(y) )#> ).]

Here the diagonal lemma seems to say that

\textbf{~(Ex)(Prf(x, <#~(Ex)(Ey)( (Prf(x,y) & This(y) )#> )#> )} \iff \textbf{~(Ex)(Ey)( (Prf(x,y) & This(y) )}
Or more succinctly, if \( \neg(\exists x)(\exists y)\left( (\text{Prf}(x,y) \& \text{This}(y) ) \right) = \text{def} = G \) then

\[
\neg(\exists x)(\text{Prf}(x, <\#G\#>) \iff \neg(\exists x)(\exists y)\left( (\text{Prf}(x,y) \& \text{This}(y) ) \right)
\]

That is if the left side is true then the right side must be true. Consequently when we prove the left side there must also be a proof of the right side. But this is only because of the absurd notion of “classical” logic that vacuous sentences are true. I have argued elsewhere (Newberry, 2016) that there are logics such that G, being vacuous, is neither true nor false, that is, the left side is true and the right side is not. The semantics is plausible, and I would argue more sensible than the “classical” one. (Newberry, 2014) No sound system based on this semantics should derive G when it derives \( \neg(\exists x)(\text{Prf}(x, <\#G\#>) \).

We can write little programs based on the logical expression on the right side of Table 1.

```
ProgramContraSelf(void)
    x := 0
    y := 0
    LOOP
        IF ( Prf(x,y) & This(y) )
            return x
        END_IF
        Get_next(x,y)
    END_LOOP
END_ProgramContraSelf
```

On the standard account this program does not halt in the standard model, but does halt in any non-standard model. Eh? So does it halt or not? In any non-standard model it halts after infinitely many steps. But there is no way to specify that we are only interested in what happens in a finite number of steps (i.e. there is no way to pin down the standard model.) This is the orthodoxy. Never mind that it does not make any sense whatsoever.

So let us try a different paradigm. If \( \neg(\exists x)(\exists y)\left( (\text{Prf}(x,y) \& \text{This}(y) ) \right) \) is \( \neg T \& \neg F \) does it mean that “ProgramContraSelf() halts” is \( \neg T \& \neg F \)? I think it is not an unreasonable
conclusion.

We know that 'Prf(x,y) & This(y)' will not be true unless y = <# ~(Ex)(Ey)( Prf(x,y) & This(y) )#>. Therefore we can simplify and write

ProgramContraSelf_2(void)
    y := <# ~(Ex)(Ey)( Prf(x,y) & This(y) )#>
    x := 0

    LOOP
        IF ( Prf(x,y) & This(y) )
            return x
        END_IF
        x := x+1
    END_LOOP
END_ProgramContraSelf

We can reason that “ProgramContraSelf_2() halts” cannot be effectively falsified. It would take an infinite number of iterations to do so. Hence it is not false. If we take the position that 'false' means “has been falsified” then the statement above it clearly not false as there is no way to falsify it. But if Prf() is sound then it is certainly not true that ProgramContraSelf() halts as a contradiction would follow. Thus “ProgramContraSelf() halts” is not true. Yet we cannot say that it is false. Given this semantics no sound decider should in fact prove that ProgramContraSelf() does not halt even though it might decide that HaltsOnItself(AntiSelf) does not.

Let us now briefly consider a program with one parameter below.

ProgHaltsOnItself(y)
    x := 0

    LOOP
        IF ( Prf(x,y) )
            return x
        END_IF
        x := x+1
    END_LOOP
END_ProgHaltsOnItself

Let us call it like this:
ProgHaltsOnItself(<#~(Ex)(Ey)(Prf(x,y) & This(y))#>)

It is false to say that the above halts. To a practical programmer it does not seem to make much difference if <#~(Ex)(Ey)(Prf(x,y) & This(y))#> is passed as a parameter or assigned inside the routine. But the point here is that in the first case <#~(Ex)(Ey)(Prf(x,y) & This(y))#> is a part of the program description. ProgramContraSelf_2() thus bears a similarity to a Quine. It basically contains a replica of itself. That makes all the difference.

* * * * *

We can expand AntiSelf()

AntiSelf() {
    if ( DoesHalt( AntiDiag, AntiDiag ) )
        loop
    else
        halt
}

DoesHalt() may choose to decide if AntiDiag( AntiDiag) halts by calling or emulating AntiDiag( AntiDiag).

DoesHalt( Program, Input ) {
    ... 
    if (Program == Input == AntiDiag) {
        AntiDiag( AntiDiag )
        return true
    }
    ... 
}

But AntiDiag( AntiDiag ) is nothing else than

AntiDiag( AntiDiag ) {
    if ( DoesHalt( AntiDiag, AntiDiag ) )
        loop
    Else
        halt
}

That is, DoesHalt() will call itself recursively with parameters AntiDiag, AntiDiag. It
will do so for ever as the only way to find out if \( \text{DoesHalt}(\text{AntiDiag}, \text{AntiDiag}) \) is to find out if \( \text{DoesHalt}(\text{AntiDiag}, \text{AntiDiag}) \). The very fact that \( \text{DoesHalt}(\text{AntiDiag}, \text{AntiDiag}) \) goes into infinite recursion means that the determination if \( \text{DoesHalt}(\text{AntiDiag}, \text{AntiDiag}) \) is true or false cannot be made. Such is the semantics of \( \text{DoesHalt}() \). Hence it is \(~(T \lor F)\). After all \( \text{DoesHalt}() \) is a predicate, and its truth values depend on its semantics. But \( \text{DoesHalt}() \) does know that \( \text{AntiSelf}() \) goes into infinite recursion, i.e. that it cannot determine if “\( \text{AntiSelf}() \) halts” is \( T \lor F \). In other words \( \text{DoesHalt}() \) knows that “\( \text{AntiSelf}() \) halts” is \(~(T \lor F)\) because it can determine that \( \text{HaltsOnItself}(\text{AntiSelf}) \) does not halt.

The argument has been made, in a similar but necessarily the same context as ours, that if \( \text{DoesHalt}() \) is sound then \( \text{DoesHalt}(\text{AntiDiag}, \text{AntiDiag}) \) must not halt, “but the algorithm cannot know this”. (Penrose, 1989, p. 65) Let us ponder this. Nothing in mathematics can be the case unless it is provable. If the algorithm “does not know it” it means that it cannot be proven. And if it cannot be proven then it cannot be the case. That is, “\( \text{AntiSelf}() \)” halts is not false. But clearly it is not true for a contradiction would occur. I was wondering how anything could possibly be the case without it being provable. Perhaps a machine not halting is some absolute Platonic fact. If so then the Liar Paradox is a proof of the existence of a Platonic universe. I do not know how anyone could seriously entertain such an idea.

* * * * *

Compare all this with Gaifman's solution of the Liar Paradox:

The following two-line puzzle will serve as our standard example.

**Line 1** The sentence on line 1 is not true.
**Line 2** The sentence on line 1 is not true.

Let us take a closer look at the failure of the line 1 sentence. The standard evaluation rule for a sentence of the form ‘The sentence written in/on ... is true’ is roughly this:

\((*)\) Go to ... and evaluate the sentence written there. If that sentence is true, so is ‘The sentence written in ... is true’, else the latter is false.

7
To get the truth-value of the negated sentence (‘The sentence written in/on ... is not true’) we should apply (*) and follow it up by applying the rule for negation (where the latter step is supposed to reverse the truth-value). In the case of the line 1 sentence, the evaluation does not terminate; the sentence sends us back to the starting point.

... The closed loop yields a non-terminating evaluation, and for this reason alone the sentence is not true.

... The conclusion that the line 1 sentence is not true reflects the realization that the straightforward implementation of (*) fails. It is expressed by using tokens different from the line 1 token, e.g., the other tokens on this page, including the one on line 2. The other tokens succeed because they are external to the loop produced by the first token. [Emphasis added]

(Gaifman, 2000, p. 3)

So much Gaifman. The token on line 1 plays a role analogical to our AntiSelf() while the token on line 2 play a role analogical to HaltsOnItself( AntiSelf ). Gaifman takes the position that different sentence tokens of the same sentence type can have different truth values. This paradigm is consistent and not entirely unreasonable. But it is also possible to take the position that that all sentence tokens of the same sentence type have the same truth value. We simply add another “layer”, e.g. “The sentence on line 1 is not true” is not true. The whole sentence is true. More on this see Newberry (2015, p.2).

We seem to think that a program either halts or not. But maybe not. Maybe there is a subtle difference between being false and not true, and maybe this subtlety comes into play just in the case of self-referential self-contradictory programs. But even if we do not embrace this controversial idea, I am still suggesting that there likely exists a program DoesHalt( ) that determines that HaltsOnItself() does not halt on AntiSelf(). It would effectively neutralize the Halting Theorem, and possibly prove all the cases when programs that are not self-referential and self-contradictory do not halt. I am aware that I have not proven the existence of such program. I am merely showing that the assumption of its existence does not lead to any obvious contradiction. But I am pretty sure it does exist.
Bibliography

Gaifman, H. (2000) Pointers to propositions. In: A. Chapuis & A. Gupta, eds. *Circularity, Definition, and Truth*, New Delhi: Indian Council of Philosophical Research: 79–121, Available at: http://www.columbia.edu/~hg17/gaifman6.pdf [Accessed 15 May 2016].

Newberry, X.Y. (2014) Generalization of the Truth-relevant Semantics to the Predicate Calculus, http://arxiv.org/ftp/arxiv/papers/1509/1509.06837.pdf

Newberry, X.Y. (2015) Reflexions on Gaifman's Solution of the Liar Paradox http://xnewberry.tripod.com/ReflexionsGaifman_2015_05_02.pdf

Newberry, X.Y. (2016) The Diagonal Lemma Fails in Aristotelian Logic, *The Reasoner* Volume 10, Number 1, http://arxiv.org/abs/1510.04393

Penrose, R. (1989) *The Emperor's New Mind*, Oxford, Oxford University Press

Copyright © X.Y. Newberry 2016
For non-commercial purposes this article may be freely copied and distributed.