the Jeffrey Shallit affair 2014 January 14

I was invited to talk about "Problems with the Halting Problem" at the University of Waterloo. Professor Jeffrey Shallit of that university was so annoyed that, on 2013 October 24, he wrote a blog titled "Eric Hehner's Fringe Computer Science". He said "Up until now, I hadn't seen too much fringe computer science. But now I have. And to make things worse, we have apparently asked the author of these fringe works to come speak at our university.". He called my work "junk", and compared me to fraudulent archaeologists, circle-squarers, and people who believe 1>2. On 2013 October 27 I replied to his blog, to which he snapped back "By the way, you forgot to compare yourself to Galileo.".

I arrived at the university to give my talk on 2013 November 28. I had never met Shallit, but I recognized him from the picture beside his blog. Just before starting my talk, I said "I'm surprised to see you here.", to which he replied "I could use a laugh.". I got no further than halfway down my first slide, having made one definition, when Shallit spoke up. He said that I had already gone wrong, and that there's no point in continuing. I replied that I am just starting to present the standard incomputability argument, and I haven't yet begun to talk about the problems with it. Shallit said that my version of the Halting Problem is nonstandard and faulty, and repeated that there's no point in continuing. I tried to continue, but Shallit would not allow me to. Other members of the audience joined the fight, all arguing against Shallit, saying that he should let me continue, and save his objections to the end. After that shaky start, having lost valuable time, I completed most of my talk and asked for questions. Shallit left.

On 2013 December 3, Shallit sent me an email saying that he had figured out where I went wrong, and how to set me straight. He included the following program and argument (I have cleaned it up a tiny bit, and restated the conclusion).

function *prints2* (*p*, *i*: **string**): **boolean**;
{ returns *true* if string *p* represents a well-formed procedure with one string input }
{ that prints 2 when executed with input *i* ; returns *false* otherwise; }
{ the function *prints2* prints nothing; *prints2* always returns a result }

procedure *q* (*s*: **string**);
begin
    if not *prints2* (*s*, *s*) **then** *print* (2)
end;

const *Q* = '**procedure** *q* (*s*: **string**); **begin if not** *prints2* (*s*, *s*) **then** *print* (2) **end**'

Assume function *prints2* has been written/programmed according to its specification/ description/comment. What is the result of executing *q* (*Q*) ?

If *q* (*Q*) prints 2 , then *prints2* (*Q*, *Q*) returns *true* , and so *q* (*Q*) does nothing.
If *q* (*Q*) does not print 2 , then *prints2* (*Q*, *Q*) returns *false* , and so *q* (*Q*) prints 2 .

This is a contradiction (inconsistency). Therefore function *prints2* cannot have been written according to its specification; *prints2* is incomputable.

In my reply to Shallit's email, I transformed this program to the program I presented at the start of my talk, showing that they are equivalent for the purpose of "proving" incomputability. Shallit's interruption of my talk was not just rude, it was also a false statement.

I agree with the first half of the concluding sentence: function *prints2* cannot have been written according to its specification. But that's not due to incomputability; it's due to an inconsistency (or self-contradiction) in the specification. As I said in my reply:

> Where the specification of *prints2* says "procedure represented by string *p* ", it means all procedures with one string input, including procedure *q* . The little argument following the listing of procedure *q* shows the inconsistency by substituting *Q* for *p* . If "incomputable" just means having an inconsistent/false specification, then *prints2* is incomputable. But it seems to me that "incomputable" doesn't just mean "inconsistent". It means that a well-defined function (i.e. one with a consistent specification) cannot be computed by a Turing-Machine-equivalent computer. And *prints2* , as specified above, is not such a function.

Shallit wrote:

> Given a program *p* and an input *i* , either program *p* prints a 2 or it doesn't. So defining a "printing function" *prints2* (*p*, *i*) to be *true* in the former case and *false* in the latter cannot possibly be "logically inconsistent".

and I replied:

> We agree that *prints2* is well-defined if we limit its domain of application to programs *p* that do not call *prints2* , neither directly nor indirectly. But what is *prints2* (*Q*, *Q*) ? Never mind how *prints2* will be written; just tell me what answer it is supposed to give. If it gives *true* , then **if not** *prints2* (*Q*, *Q*) **then** *print* (2) prints nothing, so *true* is the wrong result. If it gives *false* , then **if not** *prints2* (*Q*, *Q*) **then** *print* (2) prints 2 , so again that's the wrong result. How well-defined and logically consistent is that?

Shallit wrote:

> It is also quite different from your *shaves* example. For one thing, you never define the predicate *shaves* in any meaningful way; you "define" it by saying "such that the barber shaves the men in the town who do not shave themselves". But this does not specify what the image is on any particular pair (*a*, *b*) ! It is a specification of some property you would like *shaves* to have, not a definition of a function. When you say "Assume that *shaves* is computable", it is meaningless, because you haven't defined a function yet.

and I replied:

> Your function *prints2* is "defined" by a property you would like it to have. But the property fails to say what *prints2* (*Q*, *Q*) should be, just as my so-called "defining" property of *shaves* fails to say what *shaves* (*barber*, *barber*) is. Actually, the problem isn't under-definition; it's over-definition. Just failing to say what *shaves* (*barber*, *barber*) is, or what *prints2* (*Q*, *Q*) is, wouldn't be a problem. The problem is that *shaves* (*barber*, *barber*) must be both *true* and *false* , and that's inconsistent. Likewise *prints2* (*Q*, *Q*) must be both *true* and *false* , and that's inconsistent.

Shallit wrote:

> You could have gotten the same "logical inconsistency" by "defining" *hehner* (*x*) = *true* iff *hehner* (*x*) = *false* , and then claiming that "function *hehner* is not computable". Nobody would take this seriously.

and I replied:

> Exactly right. The barber paradox is just like that, but slightly less obvious. Apparently some people think, at least at first, that the *shaves* function is reasonably defined, until shown otherwise. And I say that *prints2* is like that, but even less obvious. It seems to be so well-defined that when the inconsistency is shown, people continue to think it is well-defined and blame the inconsistency on incomputability. I used the barber paradox to say that, like the halting function, it is well-defined if you don't ask it to apply to itself, and not well-defined if you do.

Shallit wrote:

> To see this, all you need to imagine is that *prints2* is not given by a program, but rather a call to an oracle.

My reply talked about the presentation in Shallit's book, but in essence it was:

> Suppose that *prints2* is an oracle; it gives its answers by magic. You still get the same contradiction: *q* (*Q*) prints 2 if and only if it does not print 2 . So the contradiction was not due to the computability assumption. It is an inconsistency in the specification of *prints2* .

Now I want to get rid of a distraction in Shallit's presentation of the Halting Problem. Although *prints2* is defined to say whether any procedure *p* , given any input *i* , prints 2 , the argument for incomputability uses *prints2* for only one procedure *q* , on only one input *Q* . So now let me write a simpler version of *prints2* that works on only the one procedure we're interested in, eliminating the parameters. As you will see, this simplified version supports the argument leading to incomputability just as well (or badly) as Shallit's version.

```
function prints2: boolean;
begin
        { returns true if procedure q prints 2 when executed; }
        { returns false if procedure q doesn't print 2 when executed }
end;

procedure q;
begin
        if not prints2 then print (2)
end
```

Assume function *prints2* has been written/programmed according to its specification/description/comment. What is the result of executing *q* ?

If *q* prints 2 , then *prints2* returns *true* , and so *q* does nothing.
If *q* does not print 2 , then *prints2* returns *false* , and so *q* prints 2 .

This is a contradiction (inconsistency). Therefore function *prints2* cannot have been written according to its specification; *prints2* is incomputable.

Repeating Shallit's statement, but specialized to this version:

> Either program *q* prints a 2 or it doesn't. So defining a "printing function" *prints2* to be *true* in the former case and *false* in the latter cannot possibly be "logically inconsistent".

And here's my reply:

> It's really easy to write the body of *prints2*. If *q* prints 2, then the body of *prints2* is just *prints2*:= *true* (return *true*). If *q* does not print 2, then the body of *prints2* is just *prints2*:= *false* (return *false*). There's no programming problem here. Just tell me whether *q* prints 2. If we suppose it does, we see that it doesn't; if we suppose it doesn't, we see that it does. There is a logical inconsistency in the definition of *prints2*.

I had one more email exchange with Jeffrey Shallit. On 2014 January 6 he wrote:

> I read your reply. I really think there's no point for me to continue. If you really want to insist that the assertion "program *p* prints 2 on input *i*" is not well-defined, then I don't see any possibility of a fruitful discussion. Either *p* prints 2 or it doesn't. If you deny this, you're denying basic logic, so how could more discussion resolve anything?

On the same day, I replied:

> I don't deny the law of the excluded middle. I do deny the well-definedness of "program *p* prints 2 on input *i*" if *p* can be program *q*, which calls the *prints2* program to determine whether program *q* prints 2. We agree that there's an inconsistency. You think the inconsistency is due to the assumption that *prints2* is computable. I think that the inconsistency is there even without that assumption; it's there if we let *prints2* be an oracle, and it's there if we just use the specification of *prints2* and not its code.

Shallit closed with:

> I don't see any point in meeting with you. It would just be a waste of both our time.

And I closed with:

> Agreed. Thank you for the time you have spent.

Why can't I and Jeffrey Shallit understand the Halting Problem the same way? We agree that it's not a matter of opinion, and that at least one of us is wrong. According to Shallit, I don't understand basic logic. According to me, Shallit is intelligent and capable of understanding my clear and logical arguments, but somehow he doesn't. Why not? I have a possible explanation.

Turing was a great computer scientist, a towering figure who laid the foundation of computer science. Although none of the achievements of practical computer science (hardware and software) depend in any way on Turing's proof of the incomputability of the Halting Problem, some theoretical computer science (Shallit's specialty) does depend on it. Shallit learned this proof long ago, when he was first learning computer science. All the authorities and textbooks agree on the validity of Turing's proof. If Turing's proof is wrong, the foundation of Shallit's professional world collapses; he no longer knows what is computable and what is not. Because I have strayed from the sanctioned belief, Shallit has ridiculed me by comparing me to crackpots, insulted me by calling my work "junk", and attempted to exclude me by calling me a "fringe" computer scientist and obstructing my talk.

A scientist is not called “fringe” just for making a mistake. There were several mistakes in Turing's paper discovered by contemporary scientists, causing Turing to submit a corrigendum to the next issue of the journal. Mistakes and corrections are a normal part of science. I think there's one more mistake in Turing's paper, and I am correcting it. Shallit thinks my correction is a mistake, and in our email exchange he is trying to correct me. That's good science. But in his blog, he uses ridicule and insult, and that's antiscience. He calls me “fringe”, not because he thinks I made a mistake, but because I dare to challenge something that he considers to be a sacred truth. But science has no sacred truths; it welcomes challenges to all its results. “Proof by authority” has no place in science. By welcoming challenges, science is self-correcting. By ridiculing a challenge, Jeffrey Shallit inhibits this self-correcting process, whether the challenge is correct or incorrect.

It is possible that Shallit cannot understand my argument because he has been too well indoctrinated by orthodox computer science. Another possibility is that Shallit does understand my argument, but he has too much to lose to admit it. The final possibility is that I have lost my ability to reason and my argument is wrong. I cannot know which possibility is correct.

Eric Hehner papers on halting