

# Re: Provability

---

*Source:* <http://sci.tech-archive.net/Archive/sci.math/2007-10/msg04527.html>

---

- *From:* galathaea <galathaea@xxxxxxxx>
  - *Date:* Wed, 24 Oct 2007 23:03:28 -0700
- 

On Oct 24, 9:28 pm, Marshall <marshall.spi...@xxxxxxxx> wrote:

On Oct 24, 8:58 pm, Michael Press <rub...@xxxxxxxxxxxx> wrote:

In article  
<1193251799.743613.21...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>,

Marshall <marshall.spi...@xxxxxxxx> wrote:

On Oct 24, 10:57 am, Michael Press <rub...@xxxxxxxxxxxx>  
wrote:

An algorithm is a procedure that takes input  
and  
terminates with a well-defined and asserted  
result.  
That it does terminate with the asserted  
result must be  
proven. Until we accept the proof, it is not  
an  
algorithm. The notion of right answer is not  
part of  
the definition of algorithm.

That definition is extremely narrow, and does not correspond  
to any usage of the term that I can recall in many years  
of being a programmer.

## Re: Provability

I'm not even sure I'd buy in to the "well-defined" part. Having a "probabilistic algorithm" doesn't sound like a contradiction in terms. In fact, Googling it just now it gets a lot of hits. Neither does "proven correct, proven terminating algorithm" sound redundant.

I fail to see the utility in defining an algorithm to be no more than a partial recursive function. A theorem is not a theorem until it is proven. What is your standard for implementing a method into production code?

To be fair, there is a world of difference between the standards of ordinary commercial software and the standards of rigor for a mathematical proof. In general, the former is driven by marketplace concerns, meaning whatever can be banged out most quickly that works a majority of the time. I think it was Scott McNealy that said that most software has the shelf life of a banana. Whereas math is for the ages.

I vaguely suppose that what is CS is called an algorithm is more like what in math would be called a formula.

i think the part of cs you refer  
is the engineer's cs

the infrastructurists  
who use it loosely  
because such distinctions do not help their daily work

theoretical foundations of cs  
often take the approach  
as indicated michael

this is inherently a verificationist  
or operationalist  
approach to algorithmic properties  
and mathematical properties in general

constructivists do not assign truth  
until a sentence is proven

because of the curry-howard isomorphism  
there is an isomorphism

Re: Provability

between sentences and types  
so this gives a type theory

this guarantees  
functions definable in this type theory  
are all computable

developing this as a foundation for computer science  
has evolved considerably over the years  
from rough BHK (brouwer–heyting–kolmogorov) types  
to martin–lof types  
to the more reflexive one's of artemov

constructivists in this tradition  
do not derive ullrich's dichotomy

-----  
galathaea: prankster, fablist, magician, liar

.