

Re: Python from Wise Guy's Viewpoint

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2003-11/0332.html>

From: Erann Gat (gat_at_jpl.nasa.gov)

Date: 11/03/03

Date: Mon, 03 Nov 2003 08:40:02 -0800

In article <ptg9zdzk.fsf@comcast.net>, prunesquallor@comcast.net wrote:

> gat@jpl.nasa.gov (Erann Gat) writes:
>
>> In article <znfelp34.fsf@comcast.net>, prunesquallor@comcast.net wrote:
>>
>>> gat@jpl.nasa.gov (Erann Gat) writes:
>>>
>>>>
>>>> *SATISFIES-SPEC is undefined. And would you kindly show me the adaptation
>>>> that solves the collatz problem? That encrypts text according to the
>>>> rijndael algorithm? That filters spam?*
>>>
>>> *Certainly. Oh, wait, *you* have to supply the formal specification, I
>>> just supply the algorithm that solves it.*
>>
>> *Fine, but you still have to write SATISFIES-SPEC first.*
>
> *If the specification is formal, then it can algorithmically decide
> if a putative input/output pair is indeed correct. So I'll just
> generate putative outputs and funcall your spec until I get one
> that matches.*

Ah, but now you've added an additional constraint, namely, that my spec be funcallable. Not all formal specifications can be rendered as functions. For example, "This program terminates" can be formally rendered, but not as a program.

So I say again, you must write SATISFIES-SPEC. The reason I insist on this is that you will find if you actually sit down to try to do it that it is not nearly as simple as you seem to imagine it to be.

>>> *Assuming you have a provable formal specification of the collatz
>>> problem,*
>>
>> *I'm not sure what you mean by a "provable" specification. The Collatz
>> problem is extremely simple. See
>> <http://mathworld.wolfram.com/CollatzProblem.html>*

>
> *Yep, I'm familiar with it.*
>
> *The problem with a `provable' specification is that it has shifted all*
> *the correctness burden on the specification rather than the program.*
> *So the program I write really doesn't have to do anything clever but*
> *systematically look for answers and ask the specification if they are*
> *correct. If the specification is provably correct, then the program*
> *is.*

But that only works if the specification is computable. Not all formal specifications are computable. I can render any specification uncomputable by adding the stipulation that the program must halt (or that it must not halt).

> >> *the following paper discusses a technique finds an algorithm*
> >> *that is within a factor of 5 of the fastest algorithm that provably*
> >> *implements the formal spec.*
> >
> > *That is a truly remarkable result. I have taken only a brief look at this*
> > *paper and it doesn't come across as immediately bogus, but I have a hard*
> > *time believing that it is true because if it were it seems to me that it*
> > *would provide the answer to, e.g. whether $P=NP$, whether prime factoring*
> > *really is hard, etc. etc.*
>
> *It *could*, but you'd have to supply the proof in the specification!*

That's not what the abstract says. It only says that you have to provide a formal system in which the correctness of the fastest algorithm can be proven, not the proof itself. The paper's algorithm purports to do that for you. What's more, it purports to do so in time that is within a factor of 5 of the fastest algorithm for solving the problem itself plus a (presumably very large) constant! If this were true, it would be a constructive proof that the question of whether $P=NP$ is tractable, though to actually get the answer you'd have to run the program, which could take a long time. Still, a constructive proof that $P=NP$ is decidable would be big, big news.

E.