

Re: Can a regular Turing Machine provide Protected Memory?

Source: <http://sci.tech-archive.net/Archive/sci.logic/2004-09/0316.html>

newstome_at_comcast.net

Date: 09/03/04

Date: Fri, 03 Sep 2004 03:31:24 GMT

In comp.theory Peter Olcott <olcott@worldnet.att.net> wrote:

>

> <newstome@comcast.net> wrote in message news:fjoYc.65234\$9d6.56329@attbi_s54...

>> In comp.theory Peter Olcott <olcott@worldnet.att.net> wrote:

>>>

>>> <newstome@comcast.net> wrote in message news:9D8Yc.251050\$eM2.4450@attbi_s51...

>>>> In comp.theory Peter Olcott <olcott@worldnet.att.net> wrote:

>>>>

>>>>> Well, you can't "make it moot", and there's nothing wrong with what
>>>>> I've written. Let me expand the basic argument into more basic steps:

>>>>>

>>>>> 1) You define your model of computation, and how you provide an
>>>>> answer in your model.

>>>>>

>>>>> 2) I define a way of constructing an input from a halt analyzer
>>>>> description such that the halt analyzer will not work on the
>>>>> input.

>>>>>

>>>>> 3) You provide a halt analyzer. You can use all of the information
>>>>> you want from step #2, knowing exactly how I'm going to construct
>>>>> my input to your halt analyzer. Even though you know this, you
>>>>> can **not** make a halt analyzer that will give a correct answer on
>>>>> my input.

>>>>>

>>>>> 4) I construct my input for your halt analyzer, following the
>>>>> procedure from step #2, and your halt analyzer gets the wrong
>>>>> answer.

>>>>>

>>>>> I was trying to lead you through this step by step before, but you
>>>>> never could complete even step #1 and define your model.

>>>>>

>>>>> Any time you want to try, just let me know.

>>>>>

>>>>> If you can effectively refute every combination of all of my

>>>>> methods then do it. Start with the one where I only write

>>>>> the output to the screen. If this refutation works, I can probably

sci.logic: Re: Can a regular Turing Machine provide Protected Memory?

>> > *construct the rest from this one.*
>> >
>> > *Note it must be framed as an input that no TM can possibly*
>> > *correctly process, otherwise it does not meet the burden of*
>> > *proof of proving a negative.*
>>
>> *No, it will not be *an* input that no TM can possibly correctly*
>> *process, because there is no such input. It will be a *process* for*
>> *creating an input, *given* a TM, such that this TM will not correctly*
>> *process that input.*
>>
>> *I wanted to do this step by step before, because you have a nasty*
>> *habit of reading the first few words of a posting, replying without*
>> *really thinking about it, and then ignoring the rest.*
>>
>> *Since it's Sunday and I obviously have way too much time on my hands,*
>> *I'll lay out the entire argument for you, following my 1–4 steps above*
>> *for your "output to the screen" case. If you have questions or*
>> *problems with any part of this, then ask, but please *try* to*
>> *understand it. I won't change the argument, and you don't change the*
>> *model as you try to do everytime your flaws become obvious (in other*
>> *words, stick to the "screen" model -- do not say "Oh, ok, then we'll*
>> *add the ability to inspect the state table" -- we debunk one model at*
>> *a time). Here we go:*
>>
>> *1) The "screen" is no different from a "write only output" that you've*
>> *talked about before, so we'll model that within the TM framework as*
>> *a write-only output tape. This is really easy to do: we'll just*
>> *have a TM with two tapes, and one of them (representing the*
>> *"screen") is write-only. The transition function f will look like*
>> *this:*
>>
>> $f(\text{state}, \text{tsym}) \rightarrow (\text{newstate}, \text{newtsym}, \text{tmove}, \text{outsym}, \text{outmove})$
>>
>> *where "state" is the current state, "tsym" is the current tape*
>> *symbol on the regular TM tape, "newstate" is the new state after*
>> *this transition, "newtsym" is the symbol written to the regular TM*
>> *tape, "tmove" is the movement of the regular TM tape head (we can*
>> *let this be "left", "right", or "stay still"), "outsym" is the*
>> *"screen output" symbol, "outmove" is the direction to move the tape*
>> *head on this output tape (same choices as "tmove").*
>>
>> *Notice that the transition function doesn't depend on the output*
>> *tape ("screen"), so the "screen" can't possibly affect the*
>> *computation -- this makes it write-only.*
>>
>>
>> *2) Given the above model, here's how I'll create an *input* for any*
>> *halt analyzer that runs in this model. First, the halt analyzer*
>> *has to be supplied, and the obvious way of doing that is that the*
>> *state transition function is specified. Say a table of 7-tuples is*

sci.logic: Re: Can a regular Turing Machine provide Protected Memory?

>> given that defines the "f" function given in part 1. This table
>> completely determines the actions of the halt analyzer in this
>> model, so completely defines the program. Whenever we use a
>> function name as a parameter, or a "value", it means this encoding
>> of the transition table for the function.
>>
>> It's also true that any way of correctly "executing" this
>> program/transition table is as good as any other, as long as the
>> transition function is correctly followed using the rules defined
>> by the model (defined back in step 1). So whether I build hardware
>> that actually realizes the model from step 1, or make a UTM program
>> that follows the state transition table, or implement it by using
>> lego mindstorm pieces, it really doesn't matter -- if it works in
>> *any* correct execution environment, it must work in *all*
>> *correct* execution environments, since all such environments do
>> exactly the same thing with this state transition table.
>>
>> Now keep in mind here, before you have a knee-jerk reaction to the
>> above statements, and trot out your debunked "turning the power
>> off" example, that in the end (step #4 below) I will be running
>> *exactly* the given halt analyzer in *your* model exactly. I will
>> not be changing any conditions.
>>
>> Now consider a UTM that *exactly* and *correctly* implements your
>> model. This is very easy to do -- it simply keeps track of the
>> current state, and keeps the tape contents in arrays, along with
>> the current tape position. The "code" of my UTM follows your state
>> transition table, so after every step of its execution it will be
>> in exactly the same state, with exactly the same tape contents, as
>> your machine. If your halt analyzer works in your model, it
>> clearly works in my UTM as well, since my UTM exactly implements
>> your model.
>>
>> However, my UTM does one more thing -- after the execution is
>> completed, it can look at the "write-only output tape". *Your*
>> program can't do that, because your model doesn't allow it, but my
>> UTM certainly can. So my UTM code looks like this:
>>
>> void myUTM(HA, 2ndParam) {
>> while (not in a final state) {
>> Execute HA exactly as defined by your model (given the state
>> transition table provided as the 1st parameter to myUTM), using
>> input <2ndParam, <HA, 2ndParam>>
>> }
>>
>> result = Look at the simulated output tape to see what the halt
>> analyzer output
>>
>> if (result == "Halts")
>> Loop forever;
>> else

sci.logic: Re: Can a regular Turing Machine provide Protected Memory?

>> *halt;*
>> *}*
>>
>> *Notice that myUTM is just a program, which takes a transition table*
>> *and an input to a TM as its input. The first part of myUTM (the*
>> *while loop) exactly executes HA(2ndParam, <HA,2ndParam>) following*
>> *the rules of your model. Let me repeat that last part with*
>> *emphasis: *FOLLOWING THE RULES OF YOUR MODEL*.*
>>
>> *The input for myUTM is going to be the pair <HA,myUTM>, which means*
>> *that the first part of this code exactly executes*
>> *HA(myUTM, <HA,myUTM>). This is the input that I will supply to your*
>> *halt analyzer: the machine to analyze is "myUTM", and the input for*
>> *that machine is "<HA,myUTM>". This is easy to write down, once you*
>> *supply a halt analyzer in step 3, so this defines my procedure for*
>> *producing the input that your halt analyzer can't correctly*
>> *process.*
>>
>>
>> *3) Now you get to supply whatever halt analyzer you like, claiming*
>> *that it works in your model with the write only output. Let's call*
>> *this PHA for "Peter's Halt Analyzer", and remember that it takes*
>> *two inputs, "machine" and "input".*
>>
>>
>> *4) Now I'll show you that your halt analyzer (PHA), run in your model*
>> *with the write-only output (the "screen"), will not work on the*
>> *input that I described in step 2. Notice that I don't care how*
>> *your PHA works. It is perfectly free to examine the input given to*
>> *it to see if the machine is "myUTM", since you know exactly what*
>> *that is. In other words, PHA is perfectly free to see if I'm*
>> *trying to fool it. And it can "know" when I'm trying to fool it.*
>> *It's just that it can't do anything about it. Here's the full*
>> *argument:*
>>
>> *We're running PHA(myUTM, <PHA,myUTM>) in your model, under your*
>> *conditions. Since it's your model and your conditions, it's*
>> *required to output the answer of the halt analysis to the "screen",*
>> *which must be "Halts" or "Doesn't Halt", so consider each case*
>> *separately:*
>>
>> *Case 1 ("Halts"): Remember what we said in step 2 -- the first*
>> *part of myUTM (through the while loop) runs PHA(myUTM, <PHA,myUTM>)*
>> *in *your* model. In other words, this part runs exactly the same*
>> *code, in exactly the same way, as the call to PHA. Since this*
>> *case is defined by the fact that PHA(myUTM, <PHA,myUTM>) says*
>> *"Halts" (to the "screen"), the value of "result" in myUTM will be*
>> *"Halts". This in turn makes myUTM loop for ever, so it doesn't*
>> *halt. Therefore, the answer of PHA is wrong.*
>>
>> *Case 2 ("Doesn't Halt"): Same basic argument as case 1.*

sci.logic: Re: Can a regular Turing Machine provide Protected Memory?

- >
- > *If we assume that MyUTM can really get access to the output*
- > *result of PHA, then this would result in Case 3 ("Nothing Reported").*
- > *Because what tom_usenet said (it can't tell when its being emulated)*
- > *it would not report anything for both the inner and outer invocation.*
- > *It must choose to not report anything in this case because it would*
- > *know that either answer would be incorrect.*

If it gives an answer of "Nothing Reported" any time it can't tell when it's emulated, then it must give an answer of "Nothing Reported" *ALL* the time, since it can't *ever* tell if it is being emulated.

- > *This would show that This version of PHA results in undecidability.*

I'm not sure exactly what that means, but I *think* you're conceding that this model and version of PHA can't solve the halting problem. Is that right?

- > *I would not have two tapes like you suggested. I would have one*
- > *tape, and a physical hardware output device that it accessed via*
- > *a state transition. One transition would result in "It Halts" being*
- > *written to the screen, and the other would result in "It Does Not Halt"*
- > *being written to the screen.*
- >
- > *I guess that all that you would have to do would be to append*
- > *one more state transitions out of my final states to get the results*
- > *required to make this proof work.*

Yes, that's right. You still wouldn't be able to solve the halting problem.

- > *So now we are onto my next level enhancement. We can keep*
- > *everything else the same, changing the two tapes for the two state*
- > *transitions.*
- >
- > *My next level enhancement is the UTM that tells me if there are any*
- > *state transitions out of my final state. This can be used to determine the*
- > *inner invocation from the outer invocation. If applied to your refutation*
- > *here, this method would work. I now am providing you the opportunity*
- > *to adjust your refutation accordingly.*

The refutation is almost identical. Let's say that you are going to run PHA only on a special UTM, which can give information to the program about the program itself, which we'll call a PUTM. So what you're really doing to decide if "machine" halts on "input" is running PUTM(PHA, <machine,input>).

So how is the answer provided? If you write it to the PUTM tape (and only if the PUTM tells PHA that it is running by itself, of course), then here's what we do: PUTM is just a program, like any UTM. You've even showed what some of this program would look like in an earlier

Re: Can a regular Turing Machine provide Protected Memory?

sci.logic: Re: Can a regular Turing Machine provide Protected Memory?

posting, when you tried to describe how the PUTM could look at PHA's state table and report it back. No problem.

However, I'm going to wrap **your** UTM (the PUTM) in **my** UTM -- a standard UTM, except that it reads off the tape with the result after PUTM finishes running PHA. In other words, the input we supply to your halt analyzer is the machine myUTM (just like before), and the input is now <PUTM, <PHA,myUTM>>. Exactly like in the first argument, no program can tell if it's running on an emulator -- and that includes PUTM. So PUTM can't tell how it's running -- on an emulator that is trying to trick it or not. And of course, PHA is running **by itself** on the PUTM, just as you would want it to, so it will give a result just like any other situation where it is running by itself on the PUTM. It's just that my UTM then takes your result and does something from there (the 'LoopIfHalts' functionality).

Everything else in the argument is exactly the same. This shows that adding the ability of a program to query the UTM/TM on which it is running cannot help solving the halting problem in any way. In other words, PUTM(PHA,<machine,input>) (your model, your halt analyzer, and your context is being run) doesn't work for this <machine,input>.

```
>> Summary: We defined a model with a write-only "screen". Then we
>> described how to make an input to a halt analyzer, running in the
>> "write-only screen" model. Then we showed that if you took any halt
>> analyzer in the "write-only screen" model, and produced an input as
>> described, then the halt analyzer (in the "write-only screen" model)
>> would output the *wrong* answer TO THE SCREEN.
>>
>> This means that any halt analyzer does not work on at least one input
>> in the "write-only screen" model.
>>
>> Since this works for any possible halt analyzer in the "write-only
>> screen" model, we have shown that there cannot be a halt analyzer that
>> gives the correct answer for all inputs in the "write-only screen"
>> model.
>>
>>
>> --
>>
>> That's News To Me!
>> newstome@comcast.net
>
>
--
That's News To Me!
newstome@comcast.net
```