Turing's Oracle: The Thinking Time Machine

December 23, 2003

1 Introduction

Time machines are interesting subjects. We still do not know if they would be built or they can be ever, built. Its interesting to explore the idea of machines that use time loops to peek into answers, in the future.
Time machines would provide lucrative advantages, which can never previously be accomplished, that easily. One of them, is of course, the ability to look into the stock market, in the distant future and get a good look, at the most profitable investment.
But there are other interesting questions, one can explore. That is, would a computer connected to a time machine, sound intelligent, act intelligent, be intelligent, because it can peek at consequences of the answer, before answering them. Of course, the idea is a bit strange, after all, this is the fundamentally the grandfather paradox, expressed in a different form. Right now, the concept is very abstract, in its present state and details are only sketchy. The overall goal of this document is not only, to present new computational ideas, that one can explore with time machines, but also, to explore the power of machines, with time-machine like accuracy.

2 Hind sight: 20/20?

It may seem initially that, all problem of the world, can be solved, by looking at a working magic-8 ball. But, in fact, there are even problems, that even time machines cannot solve. A simple example, is the halting program and program A that can analyze program B and tell, whether or not B will quit, even before it is executed. If the options of the output of the program are restricted to quit/ not quit, even time machines, would never be able to give an appropriate answer, that works. The machine will be stuck, in an endless loop. Now, suppose the options are quit/not quit/non-convergent? what would happen now?
So, naturally, one asks oneself, the kinds of problems, time machines could solve,

and what they, cannot solve. After some interesting conversations with Seth, it became clear that, to prove properties of such machines, it would require an accurate definition of Time machine.

3 The Setup and the Definition

Suppose, you have some problem X. You present it, at time instant 0. Now, you go to time instant k, and look at whether the problem has been solved or not solved. If it has been solved, you take the answer back to step 0, and present it as an answer. Now, the worst case, scenario is that you would have to wait, for a k, such that until all permutations of the problem have been explored. Take answer from that k, back to step 0.
Now, naturally, if there infinite permutations of scenarios to be explored, then k would be infinite and such kind of problems will never be always solved.
On the other hand, there are very many problems, even though, have a large k, they do have finite number of permutations. Now, by peeking at the answer, at the worst case of k, would be not be the most efficient way to solve anything. Interestingly, because we are able to peek into the future, we may be able to avoid, looking at possibilities that wont work. Now, the natural question is, to ask oneself, is whether we can always do things, it in the most optimal time.

3.1 Working Out The Contradiction

Now, things may look contradictory. For example, if one can look at distant future, after enumerating through very many trials and give an answer, in one trial, and avoid all the other trails, then why should one even do, those trials one doesnt work. To answer, this aspect, of this question, we are once again, required to have a clear definition of a time machine.
There is only one time loop and it is self-consistent. In a way, the past and future reinforce each other, in such a way, that the answer is some sort of, convergence of having the answer before solving it. The words like self-consistent, re-enforce, convergence are hard to be defined in very accurately, in a formal way, but eventually, we will be able to do it

3.2 Self-Consistenty

Suppose you at trail A. And you pass through trial B, and go to trial C. Now, suppose, C connects back to A. Now, for the loop to be self-consistent, if one avoids trail B, one cannot get to C, or back to A.

3.3 Trial counting

Suppose, you are in trail k, and do that trial, you will never be able to discount that trail, from your total number of trails. However, you could avoid doing that trail completely, by peeking into the future of some instant, later than k, from any time instant less than k.

4 Presenting problems to the Time machine computer

4.1 What it does and does not understand

Suppose, you give an abstract question to the machine like is this paper, blue or red and ask for an answer, the machine, by itself does not understand the question, but what it does understand is a human, who will tell the machine, either the problem has been solved correctly or not has been solved correctly. So, in a way, it tries, both scenarios, with the human, in some hidden way, and answers the question, to his satisfaction.
Of course, as I said, before, there are definitely some limitations to questions it can answer and it cannot answer and in fact, its quite interesting that there are problems, solved by everyday computers, which cannot be solved by them.
The two types of problems

4.2 Verification Problem

Verification problems are those, whose output is either solved or not solved. There is always someone, or something, that is there to give a Boolean response to the time machine computer so that its output can be verified for correctness.

4.3 Computational Problems

Computations problems are those, whose only output, is a result, at the end of the computation. There is no one, or thing, to verify the correctness of the problem.

5 Computational Problems Cannot be Solved?

Interestingly, suppose, you feed the time machine computer, x86 machine level instructions of Windows XP and ask it to run MATLAB script, which is computed by a definitive algorithm. Because, there is no fixed way, to verify the correctness of the answer, on many different levels of implementation, the machine could output any kind of garbage.
So, to get an trustable answer, suppose, you decide to feed it, a book on MATLAB and x86 instruction set, there is still no way, for that machine to correctly, verify that it understands the book. So, the thing that happens is that, unless at some level, the computational problem is not expressed in terms of a verification problem; it can never be solved successfully.

6 The Eventual Purpose

A very important problem, central to our discussion of Time-machine computer, is of course, the Turing Test. Although, it is fairly easy to turn a Turing test,

into a verification problem, that, a problem, where the computer, either passed it or did not pass it, it is still, unclear, whether, such look-ahead computers, could somehow, manage to always pass them.

6.1 Having/Not-having Memory of Tests

If one explores the idea of such a machine taking the Turing Test, one can immediately, think of 2 types of scenarios. In one case, the person, has memory of all trails to the trial, he is in, and, in the other case, where the persons memory, is only aware of things during the current trial. There is a third case, where the person, is a time-machine himself and is able, to think like it, or better than it.

6.2 Proving Properties

To prove, any key properties, it becomes necessary that one has a formulation for presenting problems to such abstract machine, and a formulation for proving important properties of such machines, as they apply to the nature of such problems, like whether it can find a solution, or not find a solution; and whether it can do it in X amount of time.
So, if we had a Time-machine computer, how would we program it? What kind of instructions would it have? These are fundamental and essential questions, at this moment, have no definitive answers, but, they could be eventually sorted out.

6.3 Programming the Machine

And at a much lower level, there is access +n [s] instruction, which accesses some state [s], nth step from now. And there is the access n [s] instruction, which accesses some state [s], the nth step before now. There is verify F that turns true, if F, has be solved correctly and returns false, if F turns false.
There are higher-level instructions, like domain, which specify the domain of the solution. And there are abstract instructions like compute instruction, which basically, computes a function.
With the loss of generality, we will assume F, could always be specified in other Turing-complete language and be run and computed, by the computer part of the machine.

$$tobefilled$$

Problems solved by the machines. Problems, which can be solved, by such a machine, are limited to problems, which can be described within such a framework. Problems, which cannot be described, are open, in some sense.