Mr G's Little Book on

The Church-Turing Thesis

or

Can machines ever be conscious and exhibit strong AI?

Preface

The "mechanics" of this booklet are a rewrite of Section 2.5 of Roger Penrose's book "Shadows of the Mind". I was inspired to pursue this theme after first reading "Gödel, Escher, Bach" by Douglas Hofstadter which itself started out as a pamphlet on Godel's Incompleness Theorem.

Penrose himself raises and demolishes 9 possible objections to the conclusion that the human mind is following a non-computable algorithm. He then hypothesises that the mind has quantum enhanced elements. Quantum computers are only briefly mentioned as a theoretical possible but these have now been developed as working machines albeit not yet very practical.

So even if Skynet is still a possibility consciousness won't spontaneously arise as the Terminator franchise suggests. That doesn't rule out the possibilities of machine consciousness – but it can't happen within the limitations of today's hard-wired computer no matter how complex. Strong Artificial Intelligence does however remains a possibility through quantum computing though I do not personally believe that.

This booklet reformulates the C-T Thesis in terms of the modern computer which is now familiar to most of us.

Artificial Intelligence

Since originally writing this, the question of AI is now a serious concept.

Weak AI does not require a consciousness machine – just that the machine is operating at a level of sophistication that it appears to display a degree of intelligence

Medical screening is about to be relegated to the machine and therefore must pass a weak AI test – I wouldn't want my bowel cancer sample decided by an idiot.
However strong AI includes the concept of consciousness which is addressed in this booklet.

The Church-Turing Thesis

Imagine you have a giant folder containing listings of every possible computer program. They can be written in any sensible computer language – BASIC would be fine. The only constraints are that

- they all start with the first line INPUT (n) – that is the program starts with inputting an integer, and
- they all end with the command STOP,

though whether the program ever gets to the last line is key to this whole booklet.

You might argue such a folder would be impossible to create but you could easily employ another program, call it GAMMA, to write every one simply by creating every possible combination of allowable commands. The fact that most of the programs will be meaningless rubbish is neither here nor there.

Number the programs C_1 C_2 C_3 ….
Specify what happens when each

program in turn has input n, by the term $C_1(n)$ $C_2(n)$ $C_3(n)$….

The general term is thus

$C_q(n)$ is what happens when the q^{th} program is fed the number n.

Now these programs, if the list is complete, will include every possible mathematics problem.

A simple one might be "Find an odd number that is the sum of n odd numbers".

We can see immediately that the program will stop when n= 1 or 3 or 5 etc. but will never stop for n = 2 or 4 or 6 – because two odd numbers will always make an even number, never another odd number.

So our poor computer program is churning away trying to find an odd number that is two other odd numbers added together and never STOPping – yet we can immediately perceive the task to be hopeless.

How did we achieve our conclusion? **We** used our brains which some might say is just another very complex computer program. Mind you, the problems won't all be that easy – some problems have taxed the minds of mathematicians for centuries and many are still unsolved.

By unsolved we mean

- we don't yet know the answer
- it has yet to be proved there is no answer.
- It might even be undecidable – that is it can be considered as either true or false within the system.

The third possibility arises through Godel's Incompleteness Theorem which scuppered the dream that mathematicians would eventually know everything. It is known there exists at least one undecidable proposition – the "continuum hypothesis – but there may exist undecidable proposition that cannot be shown to be undecidable.

Now we state clearly that whatever we mean

"No answer" equates to our program $C_q(n)$ not STOPping.

Clearly this is an unsatisfactory state of affairs because how do we ever know

the computer programme might just be about to stop. Also computer time is valuable. There are so many other things it could be doing. We can't have it churning away forever on some particular problem that has no answer. We need it just to concentrate on those problems that have a solution, so then it can churn away usefully and eventually find the answer.

So we employ a group of very clever mathematicians – and I mean real people - and they sift through all the programs C_1 C_2 C_3…and all the values of n that might be inputted (n=1 n=2 n=3 etc.) putting to one side those that they can already "prove" won't STOP.

Then one day this group of mathematicians gives you a very useful present. It is a computer program, call it **A**, that combines all their experience and knowledge.

"This program" the leader explains, "will free us to get on with something else. Just feed in the details of each program and the value to be INPUTTED and the program **A** will tell you if C **will not stop**."

Quite what **A** does if C does STOP actually need not concern us – the only condition is that, without error, **A** sifts out the non-STOPping programs so they never get activated and waste lots of valuable computing time.

In that pile will be statements like "n odd numbers can add up to an even number" so we do in fact salvage something useful from the nonSTOPPING.

Now we have **A** we just input two numbers, q the number of the C program and n the number to be INPUTTED.

Call that **A**(q,n).

So Rule (**1**) is
If **A**(q,n) STOPS then $C_q(n)$ does not stop.

Now q can have any value, so let's give it the value n.

So Rule (**2**) is
If **A**(n,n) stops then $C_n(n)$ does not stop.

Now if you've followed this far, pay particular attention because for certain you'll think a trick has been played on you when you get to the end.

As **A**(n,n) is a computer program dependent on just one input n *(that's why we set q to n so it would meet this condition)* and we have already created a list of every possible computer program requiring just a single input, so

it must already be one of the C programs!

We don't know which one so just call it k for now.

So Rule (**3**) $A(n,n) = C_k(n)$

Now n can also take any value, so why not give it the value k.

So Rule (**4**) $A(k,k) = C_k(k)$

Using Rule (**2**) with n = k

If A(k,k) stops then $C_k(k)$ does not stop

But we already know that

$A(k,k) = C_k(k)$

so we finally demonstrate

If $C_k(k)$ stops then $C_k(k)$ does not stop.

which is a pretty amazing conclusion by any standards.

Discussion

But what does this actually mean? For certain the program $C_k(k)$ does not in fact stop, but our super computer program **A** cannot ever demonstrate this. But as **we** know that $C_k(k)$ doesn't stop, we know something that **A**(k,k) doesn't know.

But **A**(k,k) was supposed to encapsulate all the methods of the most brilliant mathematicians and presumably they could eventually have worked out if $C_k(k)$ STOPped or not if they'd been specifically asked.

How can we know something so obvious and **A** not know it?

Where you go from here depends largely on your own prejudices, but the most common stated conclusion is that the human mind cannot be reduced to simple computation *(a computer program)*, because you'll always know more than "it" knows. Also no matter how sophisticated a computer program someone else creates, you'll always be able to fool it with an input that it will churn away forever on without realising it.

Conclusions

Does this have a parallel in the world of computer viruses – no matter how sophisticated the virus checking program, will there always be another virus that will defeat it?
The C-T thesis suggests to me that there will never be either the undefeatable computer virus or the infallible virus checker. Both can exploit flaws in the other which means you'll forever be paying for upgrades

The Church-Turing thesis was first formulated purely mathematically by Alonzo Church in 1926 but reformulated by Alan Turing in terms of "Turing Machines" or more simply what today we call computers. He realised that feeding computers their own program codes would expose a limitation in any program that seemingly never bothers the human brain. Because we are conscious we can always step outside the problem and effectively say, "*Ahh – you don't fool me – I see what you've done*".

Where does that consciousness arise from though? Is it from the very complexity of the human brain as the Terminator films would have us believe? That is, computers will spontaneously become conscious when a certain level of complexity is reached. It's a neat idea but one that has no mathematical basis as just demonstrated in the Church-Turing thesis.

So does consciousness come from some additional element that does not follow the rules of mathematics? That suggests the X factor somehow lies outside the normal physical universe.

rg 1st March 200

Also available in this series is

- On My TI Calculator what's the difference between Sx and σx?

  It's not what I thought for the first 40 years of the scientific calculator
- Beyond Pascal – Multinomials and Dice Throwing

  How a lower set exercise in dice throwing led to the discovery of multinomials
- Conditional Probability and Bayes Theorem

  An investigation into the pitfalls of medical screening
- Hypercomplex Numbers

  Instead of making $i^2 = -1$ as in complex numbers what if we just make $i^2 = 1$
- Propositional Calculus

  Sherlock Holmes was the great inductive detective but not infallible
- The Harmonic Triangle

  How investigating harmonic triangles led to the discovery of a universal series summation formula