School of Computer Science Theory seminar

Birmingham Friday 28th Feb 2003

Also University of Nevada at Reno 20 March 2003

The Irrelevance of Turing machines to AI
(and maybe to computers as we know them)

Aaron Sloman

http://www.cs.bham.ac.uk/~axs/

School of Computer Science

The University of Birmingham

http://www.cs.bham.ac.uk/research/cogaff/talks/#talk22

Partly based on this paper

The irrelevance of Turing machines to AI,
in Matthias Scheutz, Editor,
Computationalism: New Directions, MIT Press, 2002.
(Also at http://www.cs.bham.ac.uk/research/cogaff/),

For background, on Turing and Turing Machines, decidability, etc. see the above book, and:
http://www.turing.org.uk
http://arxiv.org/pdf/math.LO/0209332
Also give 'Turing machine', 'undecidability', 'computability', etc. to search engines.

THANKS

- To Matthias Scheutz and Achim Jung for useful discussions
- To the Leverhulme Trust for research support

To the developers of Linux
and other free, portable, reliable, software systems,
e.g. Latex, Tgif, xdvi, ghostscript, Poplog, etc.

THEMES

1. What are Turing machines?
2. Formal models of computation and mathematical results.
3. Use of limit theorems to attack AI
4. Why the attacks are irrelevant.
5. Some counter-factual history: development of computers as machines could have happened without Turing.
6. What AI needs instead of Turing machines: appropriate architectures.
7. The importance of virtual machines: and their causal powers.
8. How some virtual machines can have infinite competence (analogous to Turing machines) despite having finite implementations.

What’s this about?

A few years ago I noticed that many people who discuss AI but don’t do it seem to assume that somehow AI is based on the idea of a Turing machine.

> This was particularly true of people criticising so-called GOFAI (Good Old Fashioned AI) which makes use of various forms of computation involving construction and manipulating symbols of various sorts.
>
> An example was Roger Penrose, in his 1989 book *The Emperor’s New Mind* where he found it necessary to give a detailed account of Turing machines as part of his attack on AI.

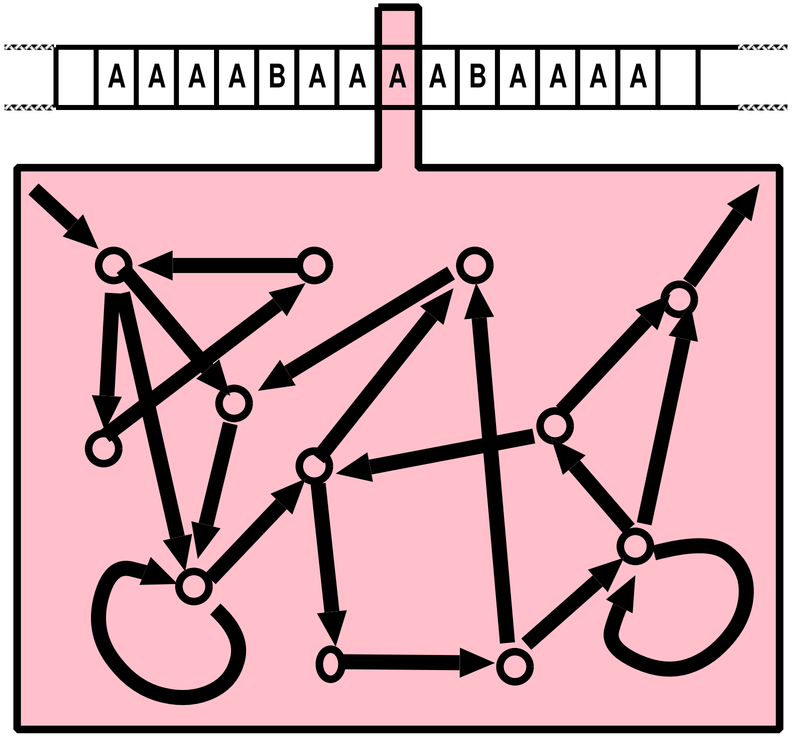
However, a survey of several of the most popular AI textbooks, revealed hardly any mention of Turing machines, although Alan Turing was rightly mentioned because of his important discussion of the possibility of intelligent machines.

So I wrote a paper (see slide 1) suggesting not only that Turing machines were not important for AI, but that historically neither Turing machines, nor any of the equivalent mathematical constructs, were important for the development of computers as we know them. This talk presents some of the ideas in the paper.

Many people who use results about Turing machines in their criticisms of AI believe that computers are completely different from brains and therefore irrelevant to neuroscience and psychology, except as tools e.g. for processing data and running models. So I tried to counter this by pointing out important ways in which brains and computers are similar, including being “sub-turing” machines, and being able to support virtual machines with infinite competence, despite finite physical size.

What's a Turing machine?

A Turing machine has a tape (infinite in one or both directions) with distinct squares on it, where each square may have one of a fixed set of marks (including blank) on it.
It has a read/write head that can examine one square on the tape, can change the mark, and can move the tape one square left or right.
It also has a set of possible states, one of which is the current state, and for each state a set of rules saying what it should do if it is in that state and the tape sensor reads particular marks. Possible actions:



Circles represent states, arrows transitions.
Arrows leading to no circle represent halting.

- Replace the current mark with another mark from the list of allowed marks.
- Move left one square, move right one square, or stay put.
- Switch to new state, or remain in old state.

A TM can be started with some initial configuration (initial state, initial tape contents, and initial read tape location). The state transition rules ("machine table") then determine a sequence of configurations in which the internal state changes and the tape changes. The changes may go on forever, or they may eventually stop.

The main reason why TMs are irrelevant

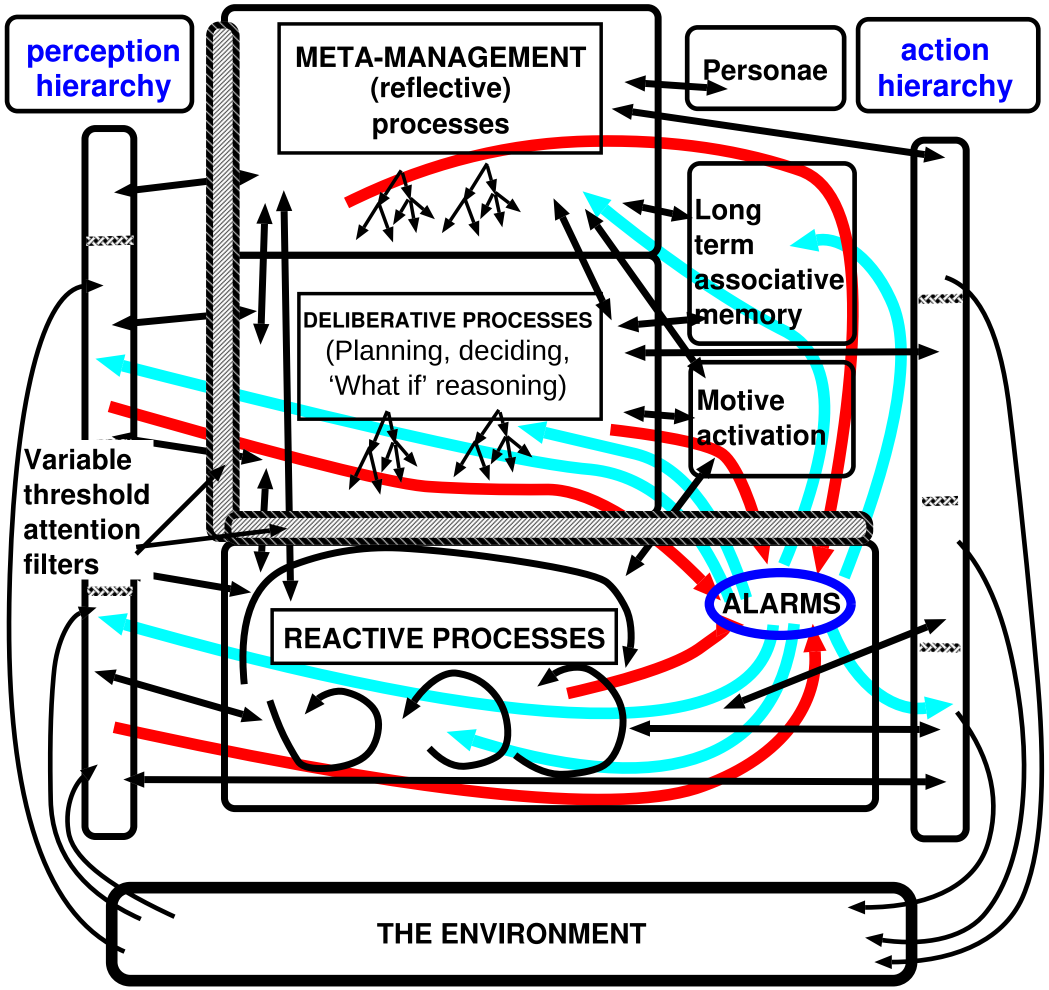
Universal Turing machines have great generality, and the general idea of a Turing machine allows a very large set of interesting algorithms to be represented. There are many deep results, e.g. concerning complexity.

However:

- **If we are trying to understand biological systems, including humans and other animals, there is no reason to believe that they have similar generality.**
  So we need to explain a collection of biological phenomena to which TMs are irrelevant.
- **Neither is there any reason to believe that the particular type of uni-process model captured in the idea of a TM is particularly helpful for information processing systems with a complex architecture, in which many different types of processes coexist and interact with one another and with the environment.**
- **Later we'll give examples of such architectures.**

Sketch of H-Cogaff, a possible architecture for human-like systems

All components shown operate concurrently.



An architecture involving huge numbers of counter-factual conditionals.

TMs are mathematical abstractions

A Turing machine specification primarily defines a class of mathematical entities, not a class of physical machines.

- The initial configuration (including tape contents and machine table specifying condition-action rules) is a mathematical structure.
- The sequence of states generated by the initial configuration is a mathematical structure (possibly infinite). (Probabilistic (stochastic)versions are also possible.)
- Various deep results, including theorems about equivalence, limit theorems, and complexity theorems were proved by Turing and others. E.g. whatever can be done using N tapes, or infinitely many tapes, can provably be done using one tape (with a cost in number of steps required).
- Gödel showed how to map discrete finite structures onto numbers, and therefore sequences of structures onto sequences of numbers. Thus all theorems about properties of TMs, and TM-based processes are equivalent to theorems about Gödel numbers and sequences of such numbers.
- Physical approximations to various kinds of TMs can be built. Insofar as a physical instance runs in a manner that "conforms to" the TM specification, the mathematical theorems will apply to it.

A “limit theorem” for physical TMs

Formal proofs of what a particular TM will or will not do may not establish results about what a physical implementation of that TM will or will not do

You cannot prove mathematically that a physical system *will* conform to a formal specification, since it is part of a larger physical environment, and a physical machine might be bombed, damaged by cosmic radiation, wear out, run out of energy, be tampered with, learn from the environment, etc.

The more interesting cases are those where it is inherent in the design that it should have rich interactions with the environment, as humans and other animals do, as will human-like AI systems.

Formal limit theorems

- Limit theorems use the fact that classes of finite discrete structures can be enumerated, i.e. mapped on to the set of positive integers.
- So all TMs, and thus their behaviours (sequences of states), can be enumerated.
- But Georg Cantor's "diagonal argument" (http://users.rcn.com/cloclo/cantdiag.html) showed that there are more infinite sequences of 0s and 1s than can be enumerated. So any enumeration of TMs to generate real numbers will leave out some real numbers (actually infinitely many).
- Therefore there are mathematical objects humans think about that TMs cannot generate or specify (e.g. the set of real numbers).
- Likewise, since there are uncountably many functions from integers to integers, and only countably many TMs, most functions from integers to integers cannot be expressed as TMs.
- Turing showed that there are "universal" Turing machines (UTMs). A UTM is one that allows any other TM to be expressed as a program on its tape. Running the UTM then simulates the running of the other TM.
- The limit theorems necessarily apply to UTMs because they are TMs.
- However, a simple physical randomiser emitting 0s and 1s need have no such restrictions: its class of possible outputs cannot be enumerated.
  Limit theorems do not apply to a machine interacting with an environment that can change it – if the environment is not equivalent to a TM.

For more on this see
http://plato.stanford.edu/entries/church-turing/ http://www.nmia.com/~soki/turing/

Varieties of formal systems

There are various classes of precisely defined specifications for formal systems which are all provably equivalent.

- TMs are one of many classes of abstractly specified formalisms for specifying logical or mathematical symbol-manipulations using “effective” methods.
- Others include
  - The class of evaluations of recursive functions (expressed in lambda calculus – A. Church)
  - The class of executions of production systems (E. Post).
  - The class of valid derivations in predicate calculus (G. Frege).
  - the class of program executions on a computer with indefinitely expandable memory.
- These different classes of abstract systems can all be proved mathematically to be equivalent, in the sense that any problem solution that can be expressed in terms of one class can also be expressed in terms of the other classes.

  This is true for every class of “effective” methods so far known.

  This is proved by showing that for everything that can occur in one class, e.g. evaluation of a lambda expression, there exists a TM or recursive function execution, or logical derivation, that models it, and *vice versa*.

Limit theorems are generally applicable

For example:

- Since Turing machines and the other formal systems are mathematically equivalent,
- it follows that something that cannot be done by any member of one such class, cannot be done any member of any other of the classes.
- So, since no TM can determine in a finite number of steps whether an arbitrary program will ever halt, no other known type of effective method can do it either.

  NOTE: Humans cannot do this either.

The Church-Turing thesis claims (roughly) that any effective formal system can be modelled in the known set of formal systems, including Turing machines.

Anti-AI Turing-Machine-based arguments

The following is a simplified version of a standard type of argument using a limit theorem to show that humans can understand and execute a procedure that cannot be expressed as a program for a UTM.

- **Consider any universal Turing machine UT$_a$ Since every program for UT$_a$ is expressible as a finite string, and finite strings can be ordered (by length and "alphabetically" if the same length), all programs for UT$_a$ can be enumerated.**
- **Consider only those programs for UT$_a$ that define a mapping from positive integers to the set $\{0\ 1\}$. They are a subset of the original set. Let the first of those programs be P_1, the second be P_2, ... P_i ... etc.**
- **We can use that enumeration to define a new function $F(n)$ as follows:**
  For any n, $F(n) = not(P_n(n))$, where $not(x)$ is 1 if $x = 0$ and is 0 if $x = 1$.
- **Then F is not defined by any P_i , as the value of $F(i)$ always differs from $P_i(i)$. So F cannot be expressed as any program for UT$_a$.**
- **Yet we know exactly what F is and for any n the value of $F(n)$ is completely defined. Therefore humans can define and think about a function from integers to $\{0\ 1\}$ not expressible as a program for UT$_a$, or any other UTM.**

There has been a huge amount of discussion of various arguments purporting to show the superiority of humans (or at least human logicians) over any "formal" system, ever since Gödel's incompleteness result was announced.

Responses to TM-based criticisms of AI

Here we give only the flavour of some of the responses.

- The limit theorems are all purely mathematical results concerning properties of abstract mathematical structures, such as the sequence of states of a TM.
- Limit theorems apply to an actual working machine only if it conforms rigidly to the mathematical specifications. But that can never be proved mathematically.
- There may be physical processes that cannot be modelled on a Turing machine.
  - Continuous processes cannot be modelled by a TM, since TMs are discrete. If a physical process has an infinitely precise measure and sweeps through an interval it covers an uncountable (non-enumerable) set of real values: a super-Turing process. If brains use continuously varying analog devices for some purpose, then AI systems can also do so.
  - If there are truly random quantum phenomena, then a physical device (which survives long enough) could randomly emit an infinite sequence of 0s and 1s. It would not be constrained by what TMs can do. If brains use randomness, so can AI systems.
- TMs are causally closed systems. Human-like AI systems (e.g. robots with sensors and motors) will constantly interact with the environment and so the limit theorems do not apply.
- Most humans, including young children, cannot understand the definition of F . Moreover, for most of the $\mathrm{P_i}$ which can be run on $\mathrm{UT_a}$, humans cannot actually compute all values of $\mathrm{P_i(n)}$, for, as n grows larger, we become less accurate, and eventually we die.
  Far from being super-Turing machines, humans are sub-Turing machines.

....Responses continued

The following points are worth noting

- **Although humans in some sense understand what F is, they cannot compute F because they can't decide which programs to include in the enumeration $\mathrm{P_i}$ – for this requires the halting problem to be solved, since otherwise we cannot tell which of the programs for UT$_\mathrm{a}$ halts with a result in $\{0\ 1\}$.**
- **It seems that even though humans cannot actually produce and run in their heads the program for function F , they can nevertheless identify the function and think about it. We do not do this extensionally by listing the input-output pairs for F , but intensionally by giving an abstract specification for F .**
- **When we have a good theory of what intensional specifications are, then perhaps we can show how to build a machine that can understand them as well as we do.**

(Thanks to Achim Jung for help in avoiding mistakes on the previous slide.)

Limit theorems and AI

Some notes on the above.

- The limit theorems concerning TMs are related to limit theorems about other powerful classes of formal systems (e.g. Gödel's incompleteness theorem).
- These results have been used by detractors of AI, e.g. John Lucas, Roger Penrose, arguing roughly
  1. Any AI system must be implementable on a Turing machine (I dispute this below).
  2. A limit theorem proves that no Turing machine can do some carefully specified task, e.g. proving that some formula X expresses a truth, thinking about some complex object.
  3. Humans can perform that task.
  4. Therefore capabilities of human minds cannot be replicated in AI systems.
- These arguments have been rebutted in many different ways over many different years.

  E.g. "humans can't do X either", "If a formula G is undecidable in formal system F1 it may be decidable in an extended system F2", etc. etc. .....

  (My 1992 review of Penrose in the AI Journal mentions other arguments.)

For now, the main point is that the assumption of a necessary connection between AI and Turing machines in premise 1 is totally unjustified: if there is some class of mechanisms more powerful than Turing machines, as the detractors claim, e.g. biological brains perhaps, or chemical computers, then AI researchers will happily use such mechanisms in building robots, etc.

(I am not claiming that biological brains are actually more powerful, merely that AI is no more defined by its tools than physics is. In both cases tools change with time.)

Another type of TM-based attack on AI: the “absurd-implementation” attack

Various philosophers (e.g. Ned Block, Selmer Bringsjord, John Searle) have attacked a ‘strong’ version of AI as follows.

1. Strong AI implies that there is some AI program AIP for a TM which, when run, necessarily creates consciousness, intelligence, emotions, etc.
2. How a TM is physically implemented does not matter.
3. So AIP could be run by having millions of Chinamen blindly cooperating in following some rules (like the transistors in a computer), or it could be run by John Searle blindly following the rules.
4. But it is “obvious” (why??) that such implementations of a TM running AIP would not have consciousness, intelligence, emotions, etc.
5. Therefore strong AI is false.

Replies to “absurd-implementation” arguments

- **Premiss 1 is false.**
- **The assumption that how something is implemented does not matter needs to be qualified.**

  For engineering purposes it does matter. The implementations proposed in these arguments would not satisfy the same collection of counterfactual conditional statements as a more conventional implementation.

  E.g. it is not true of a computer that one of its components can decide to stop cooperating, or maliciously give a wrong answer.

- **In any case it is not clear that Searle or any other human can mentally simulate an architecture with the huge amount of concurrency in H-Cogaff, with several individual components interacting causally with the environment.**

  Thus these implementations would not have the causal powers required for a mind with information autonomy (defined below). They are red herrings.

See
http://www.cs.bham.ac.uk/research/cogaff/0-INDEX00-02.html#70
http://www.cs.bham.ac.uk/research/cogaff/0-INDEX81-95.html#12

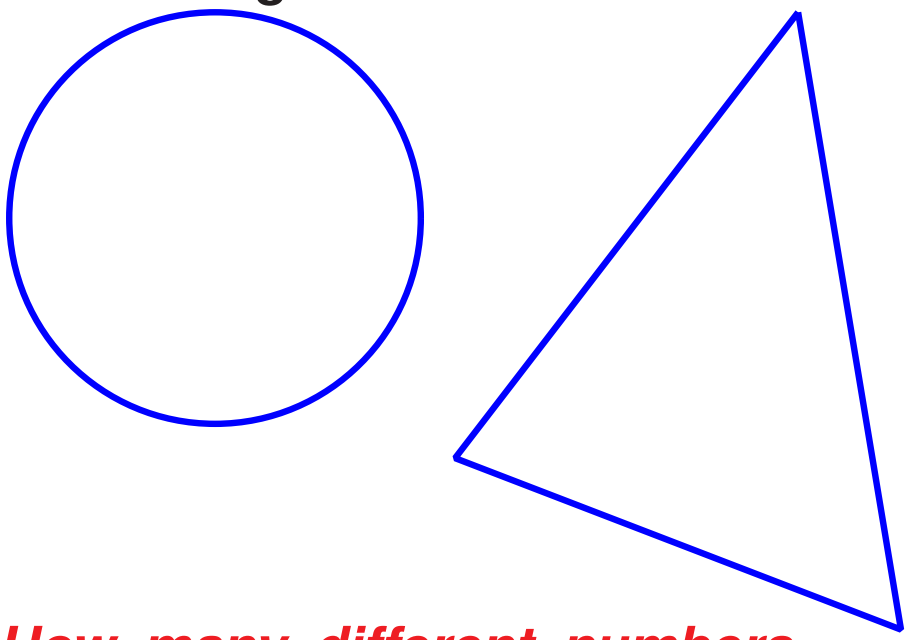
Visual reasoning in humans

Some people (e.g. Penrose) have argued that computers cannot possibly do human-like visual reasoning.

E.g. No points are common to the triangle and the circle. Suppose the circle and triangle change their size and shape and move around in the surface. They could come into contact.

If a vertex touches the circle, or one side becomes a tangent to the circle, there will be one point common to both figures. If one vertex moves into the circle and the rest of the triangle is outside the circle how many points are common to the circle and the triangle?

How do humans answer the question on the right?



How many different numbers of contact points can there be?

This requires the ability to see empty space as containing possible paths of motion, and fixed objects as things that it is possible to move, rotate and deform. Does it require *continuous* change?

Perhaps: but only in a virtual machine! To be discussed another time.

Are TMs relevant to Computers, or to AI?

It is often suggested (e.g. by critics of AI or philosophers discussing AI):

- That modern computers are somehow based on, or best understood in relation to, the notion of a Turing machine.

  More subtle versions of the claim refer to a class of concepts that are mathematically equivalent, e.g. Turing machine, lambda calculus, production system, recursive function, ...

- That the notion of computation presupposed by Artificial Intelligence research, or by Computational Cognitive Science is somehow dependent on the notion of a Turing machine.

  Or, at least, that the aims, concepts, and techniques of so-called GOFAI (good old fashioned AI) depend on the notion of a Turing machine.

- Why then do standard text books on AI (as opposed to textbooks on computer science) usually have nothing or very little to say about Turing machines, though they usually mention these:
  - Turing's 1950 paper 'Computing machinery and intelligence'
  - the Turing test described in that paper (though often inaccurately reported as a test for intelligence: Turing was far too intelligent to propose any such thing.)

Some of the confusion may have arisen because in his 1950 paper Turing used the universality of the notion of a Turing machine as an excuse for not specifying what sort of machine would be required for his thought experiment.

Uses of the mathematical notion of computation

Though not presupposed, the formal notion of computation, including the theory of TMs has practical relevance to AI and software engineering because it can apply to possible 'traces' of actual programs: traces are mathematical structures.

E.g.

- **Knowledge of limit theorems can stop people searching for ways to achieve what cannot be achieved on a computer (e.g. a general program to decide whether an arbitrary program will halt, a general test for whether a formula follows from some other formulae).**
- **More importantly, understanding of complexity results can help designers choose between different algorithms where it is important to maximise speed or minimise space requirements.**
- **Proofs of equivalence relations (or other relations) between formalisms or algorithms can eliminate wasteful debates about which is better.**
- **A mathematical analysis can detect certain bugs, e.g. due to missing conditions.**

However mathematics cannot be used to prove that any actual program running in a physical computer will work. E.g.:

- **We cannot prove that no terrorist will blow up the building, that cosmic gamma radiation will not cause a catastrophic memory fault, that no components in hard drives will fail, that no intruder will tamper with the program, etc.**
- **More interestingly you cannot prove by mathematics that the entities with which the program interacts (people, machines, the weather) will do exactly what program designers thought they would do. In particular nobody can (at present) prove theorems about how human users will interact with any machine. (Except trivially for limits imposed by the software and hardware.)**
- **Above all, sophisticated AI programs will learn from the environment.**

Fetzer's debate

[Digression]

In Sept 1988, James H. Fetzer, published a paper 'Program Verification: The Very Idea' in, Communications of the ACM.

His arguments and conclusions were partly like some of the points made here.

This led to some acrimonious discussion, part of which has been collected by Brad Cox here

http://www.virtualschool.edu/mon/SoftwareEngineering/FetzerVerificationVeryIdea.html

That collection ended with a fair amount of harmony after Fetzer clarified his position.

Related discussions can be found in this book

Timothy R Colburn (2000) *Philosophy and Computer Science* M.E.Sharpe, N.Y.

His view of virtual machines is different from that presented below.

Technology: The two historical strands

There is another way to think of computers, which has nothing to do with Turing machines or formal/mathematical models of computation: computers are devices that *do* something.

From this standpoint:
Computers are the product of two very old strands of technology.

- In the first strand machines were designed to perform *physical* control tasks: starting stopping, speeding up, slowing down or re-directing something, or generating sequences of physical events, e.g. music boxes, automatic looms.
- In the second strand, machines performed *abstract* operations on *abstract* entities, e.g. operations on or involving numbers, operations on sets of symbols to be counted, sorted, translated, etc.
  - This depended on the possibility of systematically mapping those abstract entities and abstract operations (entities in virtual machines) onto entities and processes in physical machines.
  - As the subtlety and complexity of the mapping from virtual machine to physical machine increased it allowed the abstract operations to be less and less like physical operations.
- In both strands of development, some of the machines used continuous state change (as in slide rules, analog computers and continuous governors), and others only discrete states.
- Some were deterministic others probabilistic (e.g. gambling machines).

Did the second strand require something like a theory of Turing machines?

Did the second strand of technology require designers to have something like a theory of Turing machines?

Not necessarily!

- **People designing calculators had to have a theory of numbers and numerical operations so as to ensure that all relevant numbers (usually up to a certain size) and operations were representable in their machines.**
- **People designing machines to play chess had to have a theory of the possible states of a game of chess and the constraints on permitted moves, so that they could ensure that all and only legal states were accommodated within the physical machine.**
- **People designing machines to solve problems in some other domain, e.g. solving equations, formatting documents, checking spelling, parsing sentences, checking truth-tables, sorting census data, compiling mathematical expressions into machine instructions, learning patterns in images, etc., had to have a good grasp of the particular ontology being dealt with to ensure that the machine handled all required cases.**

These requirements for designing machines to operate on abstractions were all *far more specific* than the requirement to be able to specify and understand something as general as the class of Turing machines, or the class of logical derivations.

Two kinds of abstractions: three kinds of machines

We've seen that in addition to physical machines we can have two kinds of abstract machines.

Physical processes:	Running virtual machines:	Mathematical models:
currents	calculations	numbers
voltages	games	sets
state-changes	formatting	grammars
transducer events	proving	proofs
cpu events	parsing	Turing machines
memory events	planning	TM executions

- Physical machines and the virtual machines running in them actually DO things: a calculation in a VM, or the reformatting of text in a word-processor, or a decision to turn a valve on can cause other things to change in the VM and can also cause physical events and processes – controlling machinery.
- The mathematical machines (e.g. unimplemented TMs) are abstract objects of study, but they no more act on anything in the world than numbers do, though they can help us reason about things that do act on the world, which they model, as equations can, for instance.

Two sorts of ‘running’ virtual machines

The situation is confusing if we ignore the differences between compiled and interpreted programs on computers.

- If some AI program AIP is running in a computer, as a compiled machine-code program, then it is possible that the compiled program does not go through operations of the sorts specified in the source code, e.g. because an optimising compiler has transformed the program, or because some arcane sequence of bit manipulations happens to produce the required input-output mapping.
- If AIP is stored in something close to its original form (e.g. as a parse tree) and then interpreted, the various portions of the program are causally effective insofar as they determine the behaviour produced by the interpreter: if they are changed then the behaviour changes, which will not happen if source code of a compiled program is changed.

  (Incremental compilers complicate matters, but will not be discussed here.)
- Thus if we say a program written in a batch-compiled language like C++ uses the C++ virtual machine, there is a sense in which the C++ instructions themselves have no effect at run-time, for they are replaced by machine instructions. However there could be data-structures interpreted as rules which do affect the running, e.g. rules for a game interpreted by a C++ program.
- So deciding whether a particular VM is actually running on a machine or whether it is something else that simulates it that is running, can be tricky. It all hangs on which causal interactions exist in the running VM.

Virtual machines and mental processes

- Strong AI aims not merely to replicate the input-output behaviours of a certain kind of mind but actually to replicate the internal processes, That requires making sure that we know not merely what the processes are that normally occur in such a mind, but what the causal relationships are.
- That means knowing how various possible changes in certain internal structures and processes would affect other structures and processes even if normally those changes do not occur.
- I.e. replicating mental processes in virtual machines requires us to know a great deal about the causal laws and true counter-factual conditionals (“what would have happened if”) that hold for the interactions in the system being replicated.
- Only then can we ask whether the artificially generated virtual machine truly replicates the original processes. But finding out what those laws are may be very difficult, and investigating some “what if” questions could be unethical!
- Moreover, it is not obvious that every collection of causal laws holding for human mental processes can be replicated by suitable processes running in a physical TM, since the TM implementation may not support the same set of counterfactual conditionals as some other implementation in which the higher level rules and interactions are more directly supported by the hardware.
- E.g. a neural net simulated on a serial machine typically goes through vast numbers of states which cannot occur in a parallel implementation where the nodes change state simultaneously, constraining causal interactions.

Towards a deep theory of computers and computation

A full understanding of the nature of computers – as they are now, as they have been and as they may develop – requires an understanding of what sorts of machines there are and what the space of possibilities for different sorts of machines is. I.e.

- What their functions are
- What sort of environment they operate on (e.g. purely physical environment or an environment including other information processing systems).
- Whether they are ballistic or controlled 'online'.
- Whether they have only fixed capabilities or are changeable (e.g. music box with replaceable disk)
- Whether they are physical machines or virtual machines
- What forms of representation they use and how these are manipulated
- How they are implemented - what sorts of physical implementation, and how many layers of implementation.
- Whether their operations are continuous or discrete
- Degree and kind of autonomy, e.g. whether they include internal goal generators
- How far they are self-modifying
- etc...

Machines can vary in what they operate on, use, or manipulate:

1. Matter
2. Energy
3. Information

- Scientists and engineers have built and studied the first two types of machines for centuries. Newton provided the first deep systematisation of this knowledge.
- Until recently we have designed and built only very primitive machines of the third type, and our understanding of those machines is still limited.
- Evolution 'designed' and built a fantastic variety of machines of all three types – with amazing versatility and power long before human scientists and engineers ever began to think about them.
  - Biological organisms are information-processing machines, but vary enormously in their information-processing capabilities.
  - There are myriad biological niches supporting enormously varied designs, with many trade-offs that we do not yet understand (e.g. trade-offs between cheapness and sophistication of individuals).
  - The vast majority of organisms have special-purpose information-processing mechanisms with nothing remotely like the abstractness and generality of TMs
  - A tiny subset of species (including humans) developed more abstract, more general, more powerful systems. Turing's ideas about TMs were derived from his intuitions about this aspect of human minds. But at best that's a small part of a human mind.

  Understanding all this requires us to think more about architectures than about algorithms.
  See: http://www.cs.bham.ac.uk/research/cogaff/talks/

Another kind of variation: autonomy

Machines can be more or less autonomous in various ways.

We can distinguish two main kinds of autonomy: energy autonomy and information autonomy, and sub-cases of the latter.

- **A machine may lack energy autonomy, e.g. because a person or other external agency has to turn a handle to provide energy, while having information autonomy because the machine determines how the energy is used.**

  Example: a music box, automatic loom, or mechanical calculator which requires a person to turn a handle to provide energy, while the machine decides what to do at each step.

- **A machine may lack information autonomy because a human or other external agency has to continually specify what it should do, even if it has energy autonomy.**

  Example: Someone driving a car is constantly providing control information through the use of steering wheel, brake pedal, accelerator, etc. The human does not provide energy, unlike a human riding a bicycle who provides both control information and energy.

- **Biological organisms typically exhibit both energy autonomy (except when energy levels are low and food is consumed) and information autonomy, of different kinds.**

Variations in information autonomy

Information autonomy includes different sorts of information that are useful to the organism or machine

- **Factual information**, which can be stored internally and used immediately, or later, including:
  - Information about particulars (objects, events, places, routes, times)
  - General information, universal or probabilistic, e.g. unsupported objects fall.
    Nothing is assumed about the form in which such information is stored: we still don't know much about varieties of forms of representation and their trade-offs.
- **Control information**, about what to do, including:
  - goals: immediate, future, or generic
  - plans: specific or generic ways of achieving goals
  - skills ('compiled plans').

  These may be built in (innate), learnt over a long period of training (e.g. in altricial species) or generated when needed in response to circumstances – and maybe then learnt.

Organisms vary in their information autonomy. However even when the vast majority of actions are products of internal information processing, the decisions normally make use of both currently sensed and previously acquired factual or control information developed by sensing or interacting with the environment.

Thus the information-processing system is *organism plus environment*, and unless the environment can be proved to be equivalent to a TM, theorems about TMs prove nothing about limits of biological organisms or biologically inspired AI systems.

Notions of computation

Our current notion of computation as an activity that is done by something has a number of aspects.

- **One aspect is functional and relates to what is achieved: this may be best referred to as "information processing", and has many varieties.**
- **Another aspect is structural and relates to the form of the medium in which information is stored, manipulated, transmitted, etc.**

  Another label for this is "syntactic". The use of lists, parse-trees, symbol networks, logical syntax, conditional probabilities, bit patterns, modulated wave-forms, are examples of structural, or syntactic options. ('Syntax' here does not imply discreteness.)
- **A third aspect is implementational and relates to what sort of physical machine does the information processing.**

The oldest notion of "computation" is of the first type: what humans do when they do numerical calculations. (The only one in the 1982 Concise Oxford Dictionary!).

This now seems to have been generally replaced by a notion of computation as "what we use computers to do", or more generally what information-processing machines do – and that can include brains. Neuroscientists increasingly talk about computations performed in brains.

The mathematically precise, purely formal notions of computation to which theorems about Turing machines, recursive functions and logic are relevant refer only to the second, structural aspect.

Causal aspects of computers

The first and third aspects of computation (functional and implementational) are intimately bound up with the following notions which are not applicable to the purely mathematical structures that are the subject matter of the formal theories.

- **referential (semantic) information:**
  present in running virtual machines, but not in mathematical structures.
- **causation and control**
  Involving laws of interaction and counterfactual conditionals.
- **actions performed at specific times (which can involve speed of operation),**
  Two programs for a TM can differ in how many steps they generate in sorting a list, but they cannot differ in speed. Mathematical entities don't "operate in time".
- **reliability:**
  - What a computer does can be more or less reliable, which is a feature of its causal powers, whereas the purely mathematical computation involved in sorting a list has no causal powers, and cannot be more or less reliable, just as it cannot be speeded up or slowed down.
  - Having three identical computers doing the same thing can produce a more reliable system, whereas there is no useful kind of reliability that is enhanced by writing down the same proof three times, or creating a set containing three instances of a formal computation.
  - Implementation can make a difference to computations processing information in a running virtual machine, without affecting their mathematical correlates.
    Advances in technology can improve reliability and speed of computations as we normally think of them, but cannot affect the "speed" or "reliability" of a purely mathematical object (e.g. a sequence of Gödel numbers).

What current computers are.

- Computers as we know them would have been developed even if Turing machines had never been thought of.
- They were an inevitable consequence of two very old strands of technology
  - The development of machines for automating various physical processes, for instance:
    * Windmills that turn to face the wind
    * Clocks
    * Music boxes
    * Automated looms
  - The development of machines for performing abstract operations on abstract entities, e.g.
    * doing numerical calculations
    * sorting and analysing factual information
      (e.g. census data analysed using Hollerith machines)
- Modern computers support information autonomy and discreteness, but not energy autonomy (not yet...)
- Many of the key ideas underlying the stored program computer, including the idea of self-modifying programs, were understood by Charles Babbage and Ada Lovelace a century before electronic technology made them realisable. They did not need to think about the theory of Turing machines or any of the equivalent formal mathematical systems.

Computation as what computers (and other information processors) do

Computers are used as follows:

- They are designed to acquire, store, search, compare, analyse, interpret, derive, transform, display, transmit and use information.
- They can do these things more or less quickly, more or less reliably, using more or less energy in the process, occupying more or less physical space, and interacting with more or less of the environment.
- Once programmed they often, but not always, have information autonomy: they don't need to be told what to do. However sometimes they are programmed to take external orders.
- Sometimes they have information autonomy in gaining further information from the environment, not from the designer or user: e.g. using TV cameras to find the best route from one place to another before following it.
- A computer working involves *causal* relationships of various kinds, including:
  - Controlling themselves and what happens in them
  - Controlling other things, e.g. items of machinery.
  - Being controlled by external entities, e.g. devices which they sense and respond to, or humans whose actions are perceived, analysed, interpreted, etc.
  - Learning from all the above.

Did Ada Lovelace invent AI?

See http://www.cs.yale.edu/homes/tap/Files/ada-lovelace-notes.html

It provides a selection from Ada Lovelace's notes to her translation (1843) of an article (in French) by Menabrea describing Babbage's new design. From *Ada, The Enchantress of Numbers,* by Betty Alexandra Toole, Strawberry Press, Mill Valley, CA, 1992.

- ... by the word operation, we mean any process which alters the mutual relation of two or more things, be this relation of what kind it may. This is the most general definition, and would include all subjects in the universe ...
- Again, it [the Analytical Engine] might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine . . . Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.
- ... it is likely to exert an indirect and reciprocal influence on science itself in another manner. For, in so distributing and combining the truths and the formula of analysis, that they may become most easily and rapidly amenable to the mechanical combinations of the engine, the relations and the nature of many subjects in that science are necessarily thrown into new lights, and more profoundly investigated.

By 1843 Ada Lovelace appears to have understood some of the key ideas later rediscovered by AI researchers. Her thinking was inspired by a design for a machine that did not yet exist.

Computers and brains

It used to be claimed that brains and computers had a lot in common.

Then it became fashionable to point to the differences: another way of attacking AI.

We need to do is understand that there are different levels of description and there may be deep similarities at some levels of description and huge differences at other levels.

But computers can also have huge differences at certain levels: e.g. compare current computers and the 1950s models.

Some high level similarities between brains and computers:

- Connections to sensors and motors
- Storing, using transforming information.
- Multiple forms of processing in parallel – many asynchronously.
- Huge state-space provided by using large numbers of independently switchable units, and an even larger space of possible trajectories.
- Virtual and physical machines running at the same time with mutual support.
- Both long term information stores and currently active rapidly changing information stores (e.g. sensor driven temporary stores).
- Some of what the processes operate on are external and some are internal (e.g. learning, planning, correcting mistaken beliefs, choosing a goal).

More that's common to brains and computers

The cited paper goes into more detail on these points:

- **F1. State variability**
- **F2. Laws of behaviour encoded in sub-states**
- **F3. Conditional transitions based on boolean (or fuzzy) tests**
- **F4. Referential "read" and "write" semantics**
- **F5. Self-modifying laws of behaviour**
- **F6. Coupling to environment via physical transducers (including internal environment)**
- **F7. Procedure control stack (etc.)**
  - **ability to invoke subroutines, and return to previous one.**
- **F8. Interrupt-handling and priority-handling**
- **F9. Multi-processing**
- **F10. Larger virtual data chunks**
- **F11. Self monitoring and self control**

See: http://www.cs.bham.ac.uk/research/cogaff/0-INDEX00-02.html#77

Infinite capabilities

- **Chomsky claimed (1965?) that humans have infinite linguistic competence despite finite performance.**
  E.g. there is no limit to the length of sentences in the language you understand.
- **Related kinds of infinite competence were involved in counting, thinking about longer and longer lines, thinking about transfinite sets.**
- **Many opponents thought him obviously wrong because brains are finite. He was right, but the claim needs to be formulated with care.**
- **Compare computers - define factorial.**

```
define factorial(n);
  if n == 0 then 1 else n * factorial(n-1) endif
enddefine
```

  Does a computer that has a procedure so defined available for use have infinite competence. I.e. does it include an algorithm that is defined over ALL positive integers, or only some subset?

  The answer depends on how you view the "performance limits".
  - Compare reliability: it is always relative to a class of possibilities.
  - Likewise what the computer can truly be said to implement is relative to a class of possibilities.
  - Limitations due to finite memory; number of memory slots; number of bits for addressing; size of hard drive; size of earth; size of universe...
  - We can consider ways of amending the running machine so as to overcome those limits if they threaten to get in the way, just as we can move the computer out of the path of a falling rock. This is easier if the procedure is interpreted than if it is compiled to machine code.

Architectures vs algorithms

Another way of putting all this is that a common mistake in many criticisms of AI is a failure to realise that the tools available to AI do not include only sequential algorithms but also architectures that can involve multiple concurrently and asynchronously executing processes of many kinds.

That includes the possibility of very different forms of concurrent information processing in different parts of the architecture. It may be that a small part of the system, used for symbolic reasoning, problem solving, and planning, has features that might be supported by implementation on TM.

But even if that is true, there's no reason to believe that everything is like that.

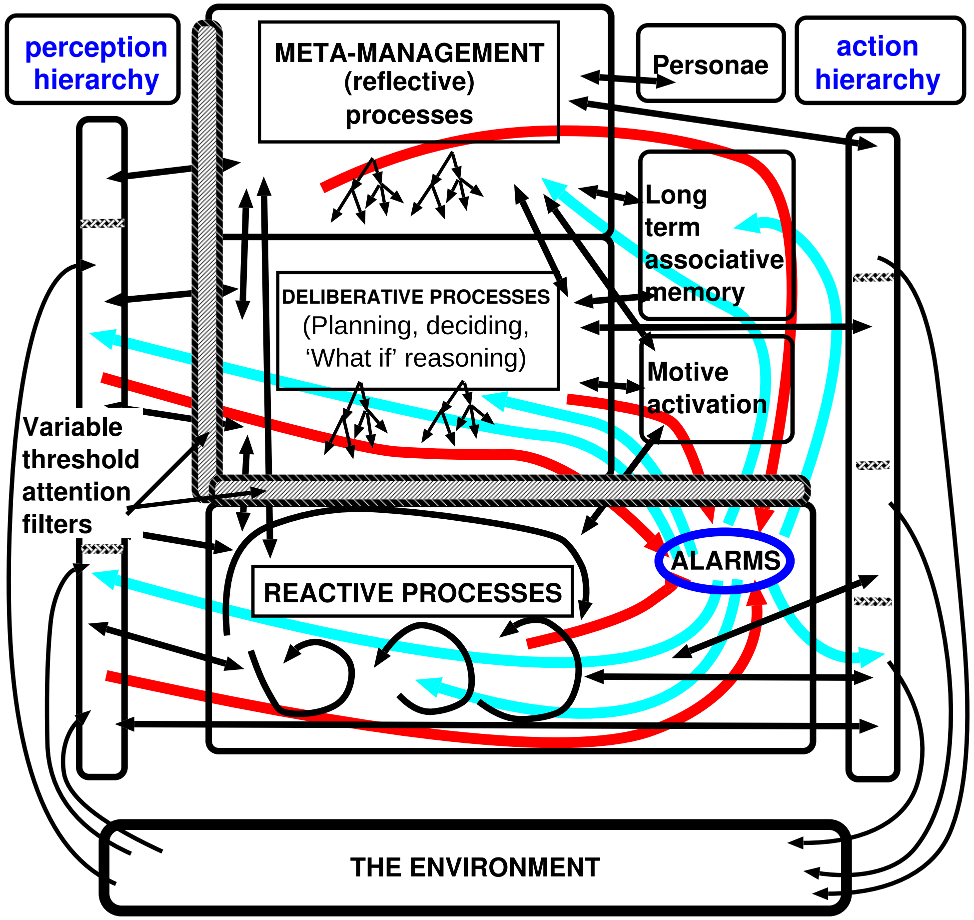
In order to decide, we'll need more comprehensive and deep theories about what kinds of information human minds process and which forms of representation and which mechanisms are suited for the task.

The point about architectures relates to the inadequacy of finite state automata (FSAs) as a framework for AI, since only one global state can exist at a time in a FSA, whereas we need multiple co-existing, interacting, concurrently active states (like most modern computing systems). (Sloman 1993: The mind as a control system)

An architecture that brains seem to implement

In various slide presentations and papers available in the CogAff project directory, I attempt to describe a notion of a class of architectures capable of explaining much that is known about different sorts of organisms.

A particularly complex special case of the class, proposed as a first draft theory of how the various aspects of the human information processing system are put together, is depicted (very abstractly) here.



Insofar as it is proposed that

1. sensors and motors are closely coupled with both the environment and the internal mechanisms and
2. there are many concurrently, asynchronously, active subsystems, some of which can interrupt and redirect others,

it is not clear that the theory of Turing machines says much about such systems.

For more details see:

http://www.cs.bham.ac.uk/research/cogaff/talks/
http://www.cs.bham.ac.uk/research/cogaff/

Architectures and attacks on Strong AI

These developments lead to the notion of "Virtual Machine Functionalism" (VMF).

- This can be contrasted with the more conventional "Atomic State Functionalism" modelled on finite state automata, or Turing machines, each with a single indivisible state at any time, with state transitions and outputs determined by current state.
  See N.Block's paper on Functionalism
  http://www.nyu.edu/gsas/dept/philo/faculty/block/papers/functionalism.html
- In contrast, VFM allows the existence of virtual machines with complex architectures, containing many concurrently active components with mutual influences, possibly connected via multiple sensors and motors to a dynamically changing environment.
- Individual components may vary in speed, may change their information content, may even modify their own behaviours, e.g. by adaptation, learning, etc.
- Instead of simple transition tables allowing one state at a time, the causal properties of the sub-mechanisms and their sub-states will involve large numbers of interacting rules and perhaps also continuous feedback loops.
- Although such VMs may be multiply realisable, these causal properties make them much harder to realise in such a way as to satisfy all the counterfactual conditionals.

This undermines the arguments of Searle, Block and others which assume that AI systems could be implemented in Searle's thought processes, or in the behaviours of millions of chinamen: those implementations would certainly fail to support all the required counterfactual conditionals. They would not be reliable realisations.

Relevance of computers to mathematics

Apart from the use of many kinds of mathematics in practical computer programs, there are several ways in which developments of computers and computer science relate to mathematics as a discipline.

1. **Contributions to metamathematics**

   Before the development of computers, logicians, mathematicians and philosophers had ideas about the nature of mathematics and mathematical reasoning.

   Developments in the theory of computation (including the theory of Turing machines) were partly inspired by and also helped contribute to meta-mathematical studies of this kind.

2. **Development of new areas of mathematics**

   The investigation of new classes of structures and processes that could occur in computational processes, led to the formation of new questions that previously had not been formulated. These include questions about complexity classes, questions about properties of algorithms, properties of data-structures, etc. In this way mathematics has been extended.

3. **Development of mathematical tools**

   Computers allowed many mathematical tasks to be automated, including checking of conjectures and proofs, searching for proofs, performing combinatorial searches, automatic differentiation, integration, and equation solving, and of course many numerical tasks.

4. **Philosophy of mathematics**

   Much of philosophy of mathematics is concerned with attempting to explain the nature of mathematical concepts, the nature of mathematical reasoning and the nature of mathematical truth. If we try to produce detailed working models of mathematical thinkers of all ages, including young children discovering properties of counting and numbers, this may lead to major new developments in philosophy of mathematics.

   Compare: http://www.cs.bham.ac.uk/research/cogaff/crp/chap8.html

AI and mathematics

I suspect that an even more important development lies in the future

As we continue to develop AI theories about the architecture of mind, especially meta-management mechanisms, and apply them to modelling development of mathematical understanding in children, we shall find that many of our ideas about mathematics and how to teach mathematics are altered in profound ways.

Conclusion

No presentation like this can settle the questions.

But I hope I have at least done something useful

- by making clearer some of the features of virtual machines that are relevant to AI (the live presentation includes a demonstration of a working virtual machine with causal powers!),
- explaining how working, causally efficacious, virtual machines, despite being abstract objects, are quite different from the abstractions of pure mathematics, which have no causal interactions
- that there are important unsolved problems about the requirements for such virtual machines, including the forms of representation they may need, the varieties of ways they can be combined in complex architectures, and whether formally equivalent implementations are causally equivalent.

The best way forward is not to spend much time discussing these rather abstract questions, but to focus most effort on designing and testing much more detailed ideas about actual working systems with human-like capabilities.

As a side-effect, that will give us new ways of thinking about these debates.

This work is supported by the Leverhulme Trust.

I thank colleagues in the Cognition and Affect project.
http://www.cs.bham.ac.uk/˜axs/cogaff.html