**In Matthias Scheutz (Ed), *Computationalism: New Directions*,
MIT Press, Cambridge, MA, pp. 87–127,**

The Irrelevance of Turing Machines to AI

Aaron Sloman
University of Birmingham
http://www.cs.bham.ac.uk/~axs/

Last corrected August 17, 2019

Contents

Abstract

The common view that the notion of a Turing machine is directly relevant to AI is criticised. It is argued that computers are the result of a convergence of two strands of development with a long history: development of machines for automating various physical processes and machines for performing abstract operations on abstract entities, e.g. doing numerical calculations. Various aspects of these developments are analysed, along with their relevance to AI, and the similarities between computers viewed in this way and animal brains. This comparison depends on a number of distinctions: between energy requirements and information requirements of machines, between ballistic and online control, between internal and external operations, and between various kinds of autonomy and self-awareness. The ideas are all intuitively familiar to software engineers, though rarely made fully explicit. Most

of this has nothing to do with Turing machines or most of the mathematical theory of computation. But it has everything to do with both the scientific task of understanding, modelling or replicating human or animal intelligence and the engineering applications of AI, as well as other applications of computers.

1 Introduction

Many people think that our everyday notion of "computation", as used to refer to what computers do, is inherently linked to or derived from the idea of a Turing machine, or a collection of mathematically equivalent concepts (e.g. the concept of a recursive function, the concept of a logical system). It is also often assumed, especially by people who attack AI, that the concepts of a Turing machine and Turing-machine computability (or mathematically equivalent notions) are crucial to the role of computers in AI and Cognitive Science.[1] For example, it is often thought that mathematical theorems regarding the limitations of Turing machines demonstrate that some of the goals of AI are unachievable.

I shall challenge these assumptions, arguing that although there is a theoretical, mathematically precise, notion of computation to which Turing machines, recursive functions and logic are relevant, (1) this mathematical notion of computation and the associated notion of a Turing machine have little or nothing to do with computers as they are normally used and thought of, (2) that although computers (both in their present form and in possible future forms if they develop) are extremely relevant to AI, as is computation defined as "what we make computers do", Turing machines are not relevant, and the development of AI did not even depend historically on the notion of a Turing machine.

In putting forward an alternative view of the role of computers and the idea of computation in AI, I shall try to clarify what it is about computers that makes them eminently suitable in principle, unlike previous man-made machines, as a basis for cognitive modelling and for building thinking machines, and also as a catalyst for new theoretical ideas about what minds are and how they work. Their relevance depends on a combination of features that resulted from two pre-existing strands, or threads, in the history of technology, both of which started hundreds, or thousands, of years before the work of Turing and mathematical logicians. The merging of the two strands and further developments in speed, memory size and flexibility were enormously facilitated by the production of electronic versions in mid 20th century, not by the mathematical theory of computation developed at the same time or earlier.

A corollary of all this is that there are (at least) two very different concepts of computation: one of which is concerned entirely with properties of certain classes of formal structures that are the subject matter of theoretical computer science (a branch of mathematics), while the other is concerned with a class of information-processing machines that can interact causally with other physical systems and within which complex causal interactions can occur. Only the second is important for AI (and philosophy of mind).

Later I shall discuss an objection that computers as we know them all have memory limits, so that they cannot form part of an explanation of the claimed *infinite* generative potential of our thought and language, whereas a Turing machine with its unbounded tape might suffice for this

[1]Turing machines are often taken to be especially relevant to so-called "good old fashioned AI" or GOFAI. This term coined by Haugeland (1985) is used by many people who have read only incomplete and biased accounts of the history of AI, and have no personal experience of working on the problems.

purpose. Rebutting this objection requires us to explain how an infinite virtual machine can be implemented in a finite physical machine.

2 Two Strands of Development Leading to Computers

Two old strands of engineering development came together in the production of computers as we know them, namely (a) development of machines for controlling physical mechanisms and (b) development of machines for performing abstract operations. e.g. on numbers.

The first strand included production of machines for controlling both internal and external physical processes. Physical control mechanisms go back many centuries, and include many kinds of devices, including clocks, musical-boxes, piano-roll mechanisms, steam engine governors, weaving machines, sorting machines, printing machines, toys of various kinds, and many kinds of machines used in automated or semi-automated assembly plants. The need to control the weaving of cloth, especially the need to produce a machine that could weave cloth with different patterns at different times, was one of the major driving forces for the development of such machines. Looms, like calculators and clocks, go back thousands of years and were apparently invented several times over in different cultures.[2]

Unlike the first strand, in which machines were designed to perform *physical* tasks, the second strand, starting with mechanical calculating aids, produced machines performing *abstract* operations on *abstract* entities, e.g. operations on or involving numbers, including operations on sets of symbols to be counted, sorted, translated, etc. The operation of machines of the second type depended on the possibility of systematically mapping those abstract entities and abstract operations onto entities and processes in physical machines. But always there were two sorts of things going on: the physical processes such as cogs turning or levers moving, and the processes that we would now describe as occurring in a *virtual machine*, such as addition and multiplication of numbers. 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.

Although the two strands were very different in their objectives, they had much in common. For instance each strand involved both discrete and continuous machines. In the first strand speed governors and other homeostatic devices used continuously changing values in a sensor to produce continuous changes in some physical output, whereas devices like looms and sorting machines were involved in making selections between discrete options (e.g. use this colour thread or that one, go over or under a cross-thread). Likewise some calculators used continuous devices such as slide-rules and electronic analog computers whereas others used discrete devices involving ratchets, holes that are present or absent in cards, or electronic switches. [3]

Also relevant to both strands is the distinction between machines where a human operator is constantly involved (turning wheels, pushing rods or levers, sliding beads) and machines where all the processes are driven by motors that are part of the machine. Where a human is involved we can distinguish cases where the human is taking decisions and feeding *control* information and the cases where the human merely provides the *energy* once the machine is set up for a task, as in a

[2]Information about looms (including Jacquard looms controlled by punched cards), Hollerith machines used for processing census data, Babbage's and Lovelace's ideas about Babbage's 'analytical engine, and calculators of various kinds can be found in *Encyclopaedia Brittanica*. Internet search engines provide pointers to many more sources. See also (Hodges 1983)

[3](Pain 2000) describes a particularly interesting analog computer, the "Financephalograph" built by Bill Phillips in 1949 used hydraulic mechanisms to model economic processes, with considerable success.

music box or some mechanical calculators. If the human provides only energy it is much easier to replace the human with a motor that is part of the machine and needs only fuel.

In short we can distinguish two kinds of autonomy in machines in both strands: energy autonomy and information or control autonomy. Both sorts of autonomy developed in both physical control systems (e.g. in factory automation) and in machines manipulating abstract information (e.g. calculators).

At first, mechanical calculators performed fixed operations on small collections of numbers (e.g. to compute the value of some arithmetical expression containing a few numerical constants, each specified manually by a human operator). Later, Hollerith machines were designed to deal with large collections of numbers and other items such as names, job categories, names of towns, etc. This made it possible to use machines for computing statistics from census data. Such developments required mechanisms for automatically feeding in large sets of data, for instance on punched cards. Greater flexibility was achieved by allowing some of the cards to specify the operations to be performed on others, just as previously cards had been used to specify the operations to be performed by a loom in weaving.

This, in combination with the parallel development of techniques for feeding different sets of instructions to the same machine at different times (e.g. changing a weaving pattern in a loom), made it possible to think of machines that modified their own instructions while running. This facility extended control autonomy in machines, a point that was apparently understood by Babbage and Lovelace long before Turing machines or electronic computers had been thought of.

A natural development of numerical calculators was production of machines for doing boolean logic, inspired by ideas of George Boole in the 19th century (and Leibniz even earlier). This defined a new class of operations on abstract entities (truth values and truth tables) that could be mapped onto physical structures and processes. Later it was shown how numerical operations could be implemented using only components performing boolean operations, leading to the production of fast, general-purpose, electronic calculating devices. The speed and flexibility of these machines made it possible to extend them to manipulate not only numbers and boolean values but also other kinds of abstract information, for instance census data, verbal information, maps, pictorial information and, of course, sets of instructions, i.e. programs.

These changes in the design and functionality of calculating machines originally happened independently of developments in meta-mathematics. They were driven by practical goals such as the goal of reducing the amount of human labour required in factories and in government census offices, or the goal of performing tasks with greater speed or greater reliability than humans could manage. Human engineering ingenuity did not have to wait for the development of mathematical concepts and results involving Turing machines, predicate logic or the theory of recursive functions, although these ideas did feed into the design of a subset of programming languages (including Lisp).

Those purely mathematical investigations were the main concerns of people like Frege, Peano, Russell, Whitehead, Church, Kleene, Post, Hilbert, Tarski, Gödel, Turing, and many others who contributed to the mathematical understanding of the *purely formal* concept of computation as some sort of philosophical foundation for mathematics. Their work did not require the existence of physical computers. In fact some of the meta-mathematical investigations involved theoretical abstract machines which could not exist physically because they were infinite in size, or performed infinite sequences of operations.[4]

---

[4] I conjecture that this mathematical approach to foundations of mathematics delayed philosophical and psychological understanding of mathematics as it is learnt and used by humans. It also impedes the development of

The fact that one of the important meta-mathematicians, Alan Turing, was also one of the early designers of working electronic computers simply reflected the breadth of his abilities: he was not only a mathematical logician but also a gifted engineer, in addition to being one of the early AI theorists (Turing 1950; Hodges 1983).

3 Combining the strands: energy and information

It was always inevitable that the two strands would merge, since often the behaviours required of control systems include numerical calculations, since what to do next is often a function of internal or external measured values, so that action has to be preceded by a sensing process followed by a calculation. What had been learned about mechanical calculators and about mechanical control systems was therefore combined in new extremely versatile information-based control systems, drawing the two strands together.

It is perhaps worth mentioning that there is a trade-off between the type of internal calculation required and the physical design of the system. If the physical design constrains behaviour to conform to certain limits then there is no need for control signals to be derived in such a way as to ensure conformity, for example. Engineers have known for a long time that good design of mechanical components of a complex system can simplify the task of the control mechanisms: it is not a discovery unique to so-called "situated" AI, but a well known general principle of engineering that one needs to consider the total system, including the environment, when designing a component. Another way of putting this is to say that some aspects of a control system can be compiled into the physical design.

Following that strategy leads to the development of special purpose control mechanisms, tailored to particular tasks in particular environments. There are many exquisite examples of such special purpose integrated designs to be found in living organisms. Evolution developed millions of varieties long before human engineers existed.

However, human engineers have also learnt that there are benefits to the design of general-purpose, application neutral, computers, since these can be produced more efficiently and cheaply if numbers required are larger, and, more importantly, they can be used after their production in applications not anticipated by the designers. Evolution appears to have "discovered" a similar principle when it produced deliberative mechanisms, albeit only in a tiny subset of animal species. This biological development also preceded the existence of human engineers. In fact it was a precondition for their existence!

Understanding all this requires unpacking in more detail different stages in the development of machines in both historical strands. This shows distinctions between different varieties of machines that help us to understand the significance of computers for AI and cognitive science.

Throughout the history of technology we can see (at least) two requirements for the operation of machines: energy and information. When a machine operates, it needs *energy* to enable it to create, change or preserve motion, or to produce, change or preserve other physical states of the objects on which it operates. It also needs *information* to determine which changes to produce, or which states to maintain. Major steps in the development of machines concerned different ways of providing either energy or information.

machines that understand numbers as humans do. Our grasp of numbers and operations on numbers is not just a grasp of a collection of formal structures but depends on a control architecture capable of performing abstract operations on abstract entities.

The idea of an energy requirement is very old and very well understood. The idea of an information requirement is more subtle and less well understood. I am here not referring to information in the mathematical sense (of Shannon and Weaver) but to an older more intuitive notion of information which could be called *control information* since information is generally potentially useful in constraining what is done. I shall not attempt to define "information" because like "energy" it is a complex and subtle notion, manifested in very many forms, applicable to many different kinds of tasks, and likely to be found in new forms in future, as previously happened with energy. So the concept is implicitly defined by the collection of facts, theories and applications in which we use it: and therefore the concept is still developing.[5]

There are many subtleties involved in specifying what information is acquired, manipulated or used by a machine (or an organism), especially as this cannot be derived unambiguously from the behaviour of the organism or the nature of its sensors. For present purposes, however, we do not need to explain in more detail how to analyse precisely what control information is used by a machine. It suffices to acknowledge that some information is required, and that sometimes designers of a machine can explain what is happening. In the present context we note the fact that one difference between machines is concerned with where the energy comes from, and another concerns where the information comes from, discussed further below.

When a human uses a machine, the degree to which either the energy or the information comes from the human or from some other source can vary. Other types of variation depend on whether the energy or the information is provided ballistically or online, or in some combination of both.

The development of water wheels, windmills, spring driven or weight driven machines, steam engines, electric motors, and many more are concerned with ways of providing energy that does not come from the user. Sometimes most of the control information comes from the user even if the energy does not. In many machines, such as cars, mechanical diggers, cranes, etc. the only energy required from the human user is that needed to convey the control information, e.g. by turning wheels or knobs, pressing buttons, or pedals, pulling or pushing levers, etc. Developments such as power-assisted steering or brakes, micro-switches and other devices reduce the energy required for supplying control information.

Sometimes the information determining what a machine should do is implicit in the physical structure of the machine and the constraints of the situation in which it operates. For instance a water wheel is built so that all it can do is rotate, though the speed of rotation is in part determined by the flow of water. In contrast, many machines are designed for use by humans who determine precisely what happens. The control information then comes from the user.

However, in general, some of the information will be inherent in the design of the machine and some will come from the environment. For instance, a windmill that automatically turns to face the wind gets its information about which way to turn from the environment.

Similar considerations apply to machines in the second strand: calculating machines. In the case of an abacus the energy to move the beads comes from the user, and most of the control information determining which beads move when and where also comes from the user. However, some of the information comes from the changing state of the abacus which functions in part as an extension of the user's memory. This would not be the case if at each step the abacus had to be disassembled and reconstructed with the new configuration of beads.

By contrast, in a primitive music box, a human may continuously provide *energy* by turning a handle while all the *control information* determining which sounds to produce next come from

[5] For more on the parallel between energy and information see the slides in this directory:
`http://www.cs.bham.ac.uk/~axs/misc/talks/`

something in the music box, e.g. a rotating cylinder or disc with protruding spokes that pluck or or strike resonating bars of different lengths. The only control the human has is whether to continue or to stop, or perhaps whether to speed up the music or slow down, depending on the construction of the music box. Some music boxes may also have a volume or tone control that can be changed while the music is playing.

Both the energy and the information required to drive a machine may be provided by a user in either an *online* or a *ballistic* fashion. If a music box accepts changeable cylinders with different tunes, the user will have control, but only ballistic control: by setting the total behaviour at the beginning. Likewise energy may be provided in a ballistic fashion, if the music box is wound up and then turned on and left to play. At the opposite extreme, playing a violin or wind instrument requires exquisite online provision of both energy and information.

The combined online provision of both energy and control information is characteristic of tools or instruments which allow humans to perform actions that are difficult or impossible for them to do unaided, because of limitations of strength, or height, or reach, or perceptual ability, or because body parts are the wrong size or shape (e.g. tweezers are often used where fingers are too big or the wrong shape) or because we cannot make the same sound as the instrument. In such cases the user is constantly in control during the operation of such a machine, providing both *energy* but also the *information* required to guide or manipulate the tool. In other machines machines most of the energy may come from some other source, while the human provides only the energy required to operate control devices, for instance when power-assisted steering reduces the amount of energy required from the user without reducing the amount of information provided. I.e. the user is still constantly specifying what to do next.

Ballistic information provision can vary in kind and degree. In the case of the music box or machine driven by punched cards the sequence of behaviours is totally determined in advance, and then the machine is allowed to run through the steps. However, in a modern computer, and in machines with feedback control mechanisms, some or all of the behaviour is selected on the basis of some tests performed by the machine even if it is running a program that was fully specified in advance. If the tests and the responses to the tests are not pre-determined, but rather produced by some kind of learning program, or by rules which cause the initial program to be modified in the light of which events occur while it is running (like an incremental compiler used interactively), then the ballistic control information provided initially is less determinate about the behaviour. It may rigidly constrain sets of possible options, but not which particular options will be selected when.

If the initial information provided to the machine makes a large collection of possible actions possible, but is not specific about the precise order in which they should be performed, leaving the machine to make selections on the basis of information acquired while behaving, then the machine is to some extent autonomous. The degree and kind of autonomy will vary.[6]

For many types of applications the control functions of the machine could be built directly into its architecture, because it repeatedly performed exactly the same sort of task, e.g. telling the time, playing a tune. This was rigid ballistic control.

For other applications, e.g. weaving cloth with different patterns, it was desirable not to have to assemble a new machine for each task. This required a separation of a fixed re-usable physical architecture for performing a class of tasks and a variable behavioural specification that could

[6]There is a "theological" notion of autonomy, often referred to as "free will", which requires actions to be non-random yet not determined by the ballistic or online information available to the agent. This was shown by David Hume to be an incoherent notion.

somehow harness the causal powers of the architecture to perform a particular task in that class.

For some of the earlier machines, the variable behaviour required continuous human intervention (e.g. playing a piano, operating a loom, or manipulating an abacus), i.e. only online control could produce variable results. Later it was possible, in some cases, to have various physical devices that could be set manually at the start of some task to produce a required behavioural sequence, and then re-set for another task, requiring a different sequence of behaviours. This was variable ballistic control. This might require setting levers or cog-wheels to some starting position, and then running the machine. In the case of the earliest electronic control systems this meant setting switches, or altering electric connections before starting the process.

At the beginning of the 19th Century, Jacquard realised that the use of punched cards could make it much easier to switch quickly between different behavioural sequences for looms. The cards could be stored and re-used as required. A similar technique used punched rolls of paper, as in player pianos. These mechanisms provided easily and rapidly specified variable ballistic control.

Later, the same general idea was employed in Hollerith card-controlled machines for analysing census data, and paper-tape controlled data-processing machines. In these cases, unlike looms, some of the ballistic control was concerned with selection of *internal* action sequences.

The alteration of such physically encoded instructions required human intervention, e.g. feeding in punched cards, or piano rolls, or in the case of some music boxes replacing a rotating disc or cylinder with metal projections.

In Babbage's design for his 'analytical engine', the use of conditionals and loops allowed the machine to decide for itself which collection of instructions to obey, permitting very great flexibility. However, it was not until the development of electronic computers that it became feasible to produce computers which, while running, could create new programs for themselves and then obey them.[7]

Machines programmed by means of punched cards had reached considerable sophistication by the late nineteenth and early twentieth century, long before electronic computers, and long before anyone had thought of Turing machines, recursive function theory, or their mathematical equivalents.

The electronic technology developed during the 20th century allowed faster, more general, more flexible, more reliable, machines to be produced, especially after the invention of transistors allowed electromechanical relays and vacuum tubes to be replaced. The advent of randomly addressable memory facilitated development of machines which could not only rapidly select arbitrary instructions to follow, but could also change their own programs easily at run time.

The process of development of increasingly sophisticated information processing systems was accelerated during the 1970s onwards, both by advances in materials science and electronic engineering, and also by the rapid evolution of new computer-assisted techniques for designing new machines and computer-controlled fabrication techniques.

In other words, the production of new improved machines for controlling physical processes accelerated the production of even better machines for that purpose. Some of this depended crucially on the second strand of development: machines for operating on abstract entities, such as numbers. The ability to operate on abstract entities was particularly important for machines to be able to change their own instructions, as discussed below. Developments in electronic technology

[7]However, many designers of electronic computers did not appreciate the importance of this and separated the memory into code and data, making it difficult to treat program instructions as data, which incremental compilers need to do. This caused problems for a number of AI language developers.

in the second half of the 20th century facilitated construction of machines which could alter their internal control information while running However the importance of this had at least partly been understood earlier: it did not depend on the idea of Turing machines, which had this capability, but in a particularly clumsy form.

3.1 Towards more flexible, more autonomous calculators

Numerical calculators, like machines for controlling physical processes, go back many centuries and evolved towards more and more sophisticated and flexible machines.

Only recently have they achieved a degree of autonomy. The earliest devices, like the abacus, required humans to perform all the intermediate operations to derive a result from some initial state, whereas later calculators used increasingly sophisticated machinery to control the operations which transformed the initial state representing a problem, to a state where the solution could be read off the machine (or in later systems printed on paper, or punched onto cards).

In the earliest machines, humans had to provide both the *energy* for making physical changes to physical components (e.g. rotating cogs), and also the *information* about what to do next. At a later date it sufficed for a human to initialise the machine with a problem and then provide the energy (e.g. by turning a handle) in a manner that was neutral between problems and did not feed additional information into the machine. Eventually even the energy for operation did not need to be supplied by a human as the machines used electrical power from mains or batteries.

3.2 Internal and external manipulations

In all these machines we can, to a first approximation, divide the processes produced by the machine into two main categories: *internal* and *external*. Internal physical processes include manipulation of cogs, levers, pulleys, strings, etc. The external processes include movements or rearrangements of various kinds of physical objects, e.g. strands of wool or cotton used in weaving, cards with information on them, lumps of coal to be sorted according to size, parts of a musical instrument producing tunes, objects being assembled on a production line, printing presses, cutters, grinders, the things cut or ground, etc.

If the internal manipulations are merely part of the process of selecting which external action to perform or part of the process of performing the action, then we can say that they are directly subservient to external actions.

However internal actions that are part of a calculation are a specially important type of action for they involve abstract processes, as discussed previously. Other abstract internal processes involve operations on non-numeric symbols and structures such as words, sentences, encrypted messages, arrays, lists, trees, networks, etc. A particularly important type of internal action involves changing or extending the initially provided information store. This gives machines considerable additional flexibility and autonomy. For instance, they may end up performing actions that were neither foreseen nor provided by the designer.

3.3 Practical and theoretical viewpoints

The requirement that a machine be able to perform abstract operations can be studied from two viewpoints. The first is the practical viewpoint concerned with producing machines that can perform useful specific tasks, subject to various constraints of time, memory requirements, cost, reliability, etc. From this viewpoint it may be sensible to design different machines for different tasks, and to give machines the powers they need for the class of tasks to which they will be applied. For this reason there are specialised machines for doing integer operations, for doing floating point operations, for doing operations relevant to graphical or acoustic applications, for running neural nets, etc. It is to be expected that the variety of different machines that are available to be combined within computers and other kinds of machinery will continue to grow.

The second, more theoretical viewpoint is concerned with questions like:

- What is the *simplest* machine that can perform a certain class of tasks?
- For a given type of machine what is the class of tasks that it can perform?
- Given two machines M1 and M2 is one of them more general, e.g. able to perform all the tasks of the other and more besides?
- Given two machines M1 and M2 are they equivalent in their capabilities: e.g. can each provide the basis for an implementation of the other?
- Is there a machine for performing abstract tasks (e.g. mathematical calculations, or logical inferences) that is *most general* in the sense that it is at least as general as any other machine that can perform abstract tasks?

From the theoretical viewpoint Turing machines are clearly of great interest because they provide a framework for investigating some of these questions, though not the only framework. If AI were concerned with finding a single most general kind of information processing capability, then Turing machines might be relevant to this because of their generality. However, no practical application of AI requires total generality, and no scientific modelling task of AI (or cognitive science) requires total generality for there is no human or organism that has completely general capabilities. There are things chimps, or even bees, can do that humans cannot and vice versa.

The mathematical applications of the idea of a Turing machine did not depend on the actual existence of such machines: they were concerned with a purely formal concept of computation. However it is possible in principle to build a Turing machine although any actual physical instance must have a finite tape if the physical universe is finite.

We can now see Turing machines as just one of a class of machines that are capable of performing either the task of controlling a physical system or of performing abstract operations, or of using one to do the other. Their most important single characteristic is the presence of an unbounded tape, but that is possible only if they are treated as mathematical constructs, for physical machines will always have a bounded tape.

However, that unique feature cannot be relevant to understanding human or animal brains since they are all finite in any case. No human being has a memory that has unlimited capacity like a Turing machine's tape. Even if we include the external environment, which can be used as an extension of an individual's memory, anyone who has written or bought many books or who has created many computer files knows that as the total amount of information one records grows the harder it becomes to manage it all, to find items that are relevant, and even to remember that you have some information that is relevant to a task, let alone remember where you have put it. There is no reason to believe that humans could manage unlimited amounts of information if provided with an external store of unlimited capacity, quite apart from the fact that we live only for a finite time.

In a later section, we'll consider the argument that these limitations of human beings are merely *performance* limitations, and that we really do have a type of infinite, or at least unbounded, *competence*. It will be shown that analogous comments can be made about conventional computers which do not have the unbounded memory mechanism of a Turing machine.

Having previously shown that the development of computers owed nothing to the idea of a Turing machine or the mathematical theory of computation, we have now given a negative answer to the question whether Turing machines, viewed as simply a special type of computer, are required for modelling human (or animal) minds because the unbounded tape of a turing machine overcomes limitations of more conventional computers.

Turing machines, then, are irrelevant to the task of explaining, modelling or replicating human or animal intelligence, though they may be relevant to the mathematical task of characterising certain sorts of esoteric unbounded competence. However computers have features that make them relevant which do not depend on any connection with Turing machines, as will now be shown.

4 Computers in engineering, science and mathematics

The features of computers that grew out of the two strands of development made them powerful and versatile tools for a wide variety of tasks which can be loosely classified as engineering, science and mathematics. The notion of a Turing machine and related logical and mathematical notions of computation are only indirectly relevant to most of these. In fact, as explained above, many of the applications were being developed before the time of Turing. AI overlaps with all of these application areas in different ways. I shall make a few comments on the relevance of computers to all these areas before going on to a more detailed analysis of the relevance of computers to AI and cognitive science. However it will help to start with an analysis of their general relevance to engineering and science.

4.1 Engineering and scientific applications

Most of the features of the new calculating and controlling engines (manipulation of physical objects and manipulation of abstract entities such as numbers or symbols) are equally relevant to a variety of different application domains: industrial control, automated manufacturing systems, data-analysis and prediction, working out properties of complex physical systems before building them, information management in commercial, government and military domains, many varieties of text processing, machine interfaces to diverse systems, decision support systems and new forms of communication. These applications use different aspects of the information manipulating capabilities described above, though with varying proportions and types of ballistic and online control, and varying proportions of physical manipulation and manipulation of abstract entities. None of this had anything to do with Turing machines.

In addition to practical applications, computers have been enormously relevant in different ways to science construed as the attempt to understand various aspects of reality. For instance they are used:

- to process numerical and other data collected by scientists,
- to control apparatus used in conducting experiments or acquiring data,
- to build working models capable of being used as explanations,
- to make predictions that can be used to test scientific theories.

For some of these uses the ability of computers to control devices which manipulate physical objects are particularly relevant, and for others the ability to manipulate abstractions such as numbers, laws, hypotheses are more relevant.

4.2 Relevance to AI

The very features that made computers relevant to all these engineering applications, and to science in general, also make them relevant to both the scientific aims of AI and the engineering aims.

The scientific aims of AI include understanding general features of both natural and artificial behaving systems, as well as modelling and explaining a wide variety of very specific naturally occurring systems, for instance, different kinds of animal vision, different kinds of animal locomotion, different kinds of animal learning, etc.

Since the key features of such natural systems include both being able to manipulate entities in the environment and being able to manipulate abstract entities, such as thoughts, desires, plans, intentions, theories, explanations, etc, the combined capabilities of computers made them the first machines suitable for building realistic models of animals.

Moreover, the tasks of designing, extending and using these capabilities of computers led to development of a host of new formalisms and concepts relevant to describing, designing and implementing information processing mechanisms. Many of these are relevant to the goals of AI, and will be described below.

The engineering aims of AI include using computers to provide new sorts of machines that can be used for practical purposes, whether or not they are accurate models of any form of natural intelligence. These engineering aims of AI are not sharply distinguished from other types of applications for which computers are used which are not described as AI. Almost any type of application can be enhanced by giving computers more information and more abilities to process such information sensibly, including learning from experience. In other words almost any computer application can be extended using AI techniques.

It should now be clear why computers are relevant to all the different sub-domains of AI dealing with specific aspects of natural and artificial intelligence, such as vision, natural language processing, learning, planning, diagrammatic reasoning, robot control, expert systems, intelligent internet agents, distributed intelligence, etc. – they all have some combination of control of physical processes and abstract information manipulation processes, tasks for which computers are better than any pre-existing type of machine.

It is noteworthy that computers are used by supporters of all the rival "factions" of AI adopting different sorts of designs, such as rule-based systems, logicist systems, neural nets, evolutionary computation, behaviour-based AI, dynamical systems, etc.

Thus there is no particular branch of AI or approach to AI that has special links with computation: they all do, although they may make different use of concepts developed in connection with computers and programming languages. In almost all cases, the notion of a Turing machine is completely irrelevant, except as a special case of the general class of computers.

Moreover, Turing machines are not so relevant intrinsically as machines that are designed from the start to have interfaces to external sensors and motors with which they can interact online, unlike Turing machines which at least in their main form are totally self contained, and are designed primarily to run in ballistic mode once set up with an initial machine table and tape configuration.

4.3 Relevance to mathematics

The relevance of computers to mathematics is somewhat more subtle than the relevance to other scientific and engineering disciplines. There are at least three types of development that link mathematics and computing:

(a) *More mathematics:* using abstract specifications of various kinds of (abstract) machines and the processes they can support, in order to define one or more new branches of mathematics, e.g. the study of complexity, computability, compressibility, various properties of algorithms, etc. This is what a lot of theoretical computer science is about. (Some of this investigates machines that could not be built physically, e.g. infinite machines, and types of machines that might be built but have not, e.g. inherently probabilistic machines.)

(b) *Metamathematics:* using an abstract specification of a type of machine as an alternative to other abstract specifications of types of mathematical objects and processes (recursive functions, Post productions, axiomatic systems, etc.), and then exploring their relationships (e.g. equivalence), possibly to clarify questions in the philosophy of mathematics.

(c) *Automatic theorem proving or checking:* using computers as tools to help in the discovery or proof of theorems, or the search for counter-examples. This process can be more or less automated. At one extreme, computers are programmed to do large numbers of well defined but tedious operations, e.g. examining very large sets. At another extreme, the computer may be fairly autonomous, taking many decisions about which steps to try in a context sensitive manner and possibly as a result of learning from previous tasks. AI work on theorem proving tends towards the latter extreme. It may also allow human interaction, such as the communication that happens between human mathematicians when they collaborate, or when one teaches another. This sort of mathematical application could build on general AI research on intelligent communication.

Although mathematical explorations of types (a) and (b) involve ideas about computation, it often does not matter whether physical computers exist or not, for they are not needed in those explorations. Many of the important results, for instance Gödel's undecidability result, were achieved before working computers were available. (Quantum computers might also be relevant to mathematical investigations of types (a) and (b) even if they turn out to be impossible to build as practically useful physical devices.) By contrast work of type (c) depends on the use of working computers.

The distinction between (a) and (b) is not yet very clear or precise, especially as (a) subsumes (b)! Neither is there a very sharp division between the meta-mathematical use of the notion of computation in (b) and the AI uses in connection with designing theorem provers, reasoners, etc.

Ideas about Turing machines and related theoretical "limit" results on computability, decidability, definability, provability, etc. are relevant to all these kinds of mathematical research but are marginal or irrelevant in relation to most aspects of the scientific AI goal of trying to understand how biological minds and brains work, and also to the engineering AI goals of trying to design new useful machines with similar (or greater) capabilities. The main relevance of the limit results arises when researchers set themselves goals which are known to be unachievable e.g. trying to design a program that will detect infinite loops in any arbitrary program.

The metamathematical ideas developed in (b) are relevant to the small subset of AI which is concerned with *general* (logical) reasoning capabilities or modelling *mathematical* reasoning.

By contrast, the new mathematical techniques of type (a) which were developed for analysing properties of computational processes such as space and time complexity and for analysing relationships between specifications, designs and implementations, are all equally relevant both to AI and to other applications of computers.

One important feature of Turing machines for mathematical or meta-mathematical research of types (a) and (b) is their universality, mentioned previously. By showing how other notions of mathematical reasoning, logical derivation, or computation as an abstract mathematical process, could all be mapped onto Turing machines it was possible to demonstrate that results about mathematical limitations of Turing machines could not be overcome by switching to any of a wide range of alternative formalisation. It also meant that analyses of complexity and other properties of processes based on Turing machines could be carried over to other types of process by demonstrating how they were implementable as Turing machine processes.[8]

This kind of mathematical universality may have led some people to the false conclusion that any kind of computer is as good as any other provided that it is capable of modelling a universal Turing machine. This is true as a mathematical abstraction, but misleading, or even false when considering problems of controlling machines embedded in a physical world.

The universality of Turing machines was mentioned by Turing in his 1950 article as a reason for not discussing alternative digital mechanisms. In part that was because he was considering a question-answering task for which there were no time constraints, and where adding time constraints would produce no interesting differences, since only qualitative features of the behaviour were of interest.

Human intelligence, however, is often *precisely* concerned with finding good solutions to problems quickly, and speed is central to the success of control systems managing physical systems embedded in physical environments. Aptness for their biological purpose, and not theoretical universality, is the important characteristic of animal brains, including human brains. What those purposes are, and what sorts of machine architectures can serve those purposes, are still open research problems (which I have discussed elsewhere), but it is clear that time constraints are very relevant to biological designs: speed is more biologically important than theoretical universality.

This section on the mathematical applications of ideas of computation was introduced only in order to get them out of the way, and in order to provide a possible explanation for the wide-spread but mistaken assumption that notions such as Turing machines, or Turing computability are central to AI. (This is not to deny that Turing was important to AI as an outstanding engineer who made major contributions to the development of practical computers. He was also important as one of the earliest AI theorists.)

4.4 Information processing requirements for AI

For the mathematical and meta-mathematical investigations mentioned above, the formal notions of computations were central. By contrast, for the non-mathematical, scientific and engineering goals of AI, the important point that was already clear by about the 1950s was that computers provided a new type of *physically implementable* machine with a collection of important features discussed in previous sections and analysed in more detail below.

These features were not defined in relation to recursive functions, logic, rule-formalisms, Turing machines, etc. but had a lot to do with using machines to produce and control sophisticated internal and external behaviour with a speed and flexibility that was previously impossible for man-made machines, although various combinations of these abilities were to be found in precursors to modern computers. Moreover some of the mathematically important features of Turing machines are irrelevant to animal brains.

[8] It is possible that some formalisms that cannot be manipulated by Turing machines, e.g. formalisms based on continuously varying geometric shapes, will turn out to be relevant to goals of AI, refuting the claimed universality of Turing machines, but that will not be discussed here.

However, I shall identify two related features that are relevant, namely (i) the ability to chunk behaviours of varying complexity into re-usable packets, and (ii) the ability to create and manipulate information structures that vary in size and topology.

A Turing machine provides both features to an unlimited degree, but depends on a linear, indefinitely extendable tape, whose speed of use is inherently decreased as the amount of information thereon increases. However, for an animal or robot mind, it is not clear that *unlimited* size of chunks or variability of structure is useful, and the cost of providing it may be excessive. By contrast computers with random access memory provide uniform speed of access to a *limited* memory. Brains probably do something similar, though they differ in the details of how they manage the trade-off.

Although humans do not have the same generality as Turing machines in their mathematical and symbolic reasoning powers, nevertheless we do have certain kinds of generality and flexibility, and I shall try to explain below how computers, and also brains, can provide them. Turing machines provide much more generality but do so in a fashion that involves such a heavy speed penalty in any working physical implementation, because of the need for repeated sequential traversal of linear tapes, that they seem to be worthless for practical purposes. It appears that brains, like computers, sacrifice total generality in favour of speed in a large class of behaviours.

4.5 Does AI require the use of working machines?

A possible source of confusion is the fact that for some of the theoretical/scientific purposes of AI (cognitive science) the actual construction of computers did not matter except as a catalyst and test-bed for ideas: the most important effect of development of computers for advances in AI and cognitive science was in the generation of ideas about how information manipulating engines might be implemented in physical mechanisms. This makes scientific AI superficially like meta-mathematics, but the similarity is deceptive, since the goals are different.

For the engineering purposes of AI, working physical machines are, of course, required. They are also needed as an aid to AI theorists, since the models being considered have grown increasingly complex, and too difficult to run in our heads or on paper. But that is like the relevance of computers to theoretical chemistry, astronomy, and other sciences. Computers are tools for managing complex theories. This has nothing specific to do with cognitive science or AI.

5 Eleven important features of computers (and brains)

More important than the use of computers as cognitive aids to analysing complex theories was the way we came to understand the features needed in computers if they were to be useful. Those features gave us new ideas for thinking about minds.

However, those features are not specific to what we now call computers: animal brains had most of these features, or similar features, millions of years earlier, and probably lots more that we have not yet thought of, and will later learn to emulate, using either computers or new kinds of machines.

There are (depending how we separate them out) about 11 major features, which I shall list below. Features F1 to F6, which I have labelled “primary features”, are typically built in to the digital circuitry and/or microcode of computers and have been common to low-level virtual machine architectures since the 1960s (or earlier), though details have changed.

The remaining features, labelled as "secondary" below, depend very much on software changing the higher level virtual machine implemented in the basic lower level machine. The need for those extra features to be added was driven both by AI requirements and by other software engineering requirements.[9]

Besides the primary and secondary features of computers as control systems listed below, there are additional features that are added through the design and implementation of various high level languages, along with compilers, interpreters, and software development tools. These extend the variety of virtual machines available. However, these split into several different families suited to different application domains, and will not be discussed here.

F1. State variability: having very large numbers of possible internal states, and even larger numbers of possible state transitions.
Both of these are consequences of the fact that computers have large numbers of independently switchable components, like brains. N 2-valued components can be in 2^N possible states, and can support the square of that number of possible one-step state transitions: this gives huge flexibility in coping with varied environments, goals and forms of learning.

Of course, it is not enough that there are large numbers of components that can change their state. That is equally true of thunderclouds. It is important that the switchable components are controlled in a principled way, depending on the system's current tasks and the available information. It is also important that those sub-states can be causally connected in a principled way to various effects, both within the machine and in external behaviour. This depends on the next three features. below.

F2. Laws of behaviour encoded in sub-states: having laws of behaviour determined by parts of the internal state.
The behaviour of any physical object is to some extent controlled by its sub-states: e.g. how a rock or a bean-bag rotates if thrown into the air will depend on its shape and the distribution of matter within it, which may change in the bean-bag. However computers have far more independently switchable persistent sub-states and their effects can be made more global: a CPU is an influence-amplifier. All this depends on the fact that a stored program typically includes components which have procedural semantics and whose execution generates state transitions in accordance with the program, e.g. using sequencing, conditionals, jumps, procedure invocation (if supported; see secondary feature F7 below), etc. The effects can propagate to any part of the system.

It is important, however, that not all stored information states are concurrently active, as occurs when a large number of different forces are simultaneously acting on a physical object whose behaviour is then determined by the resultant of those forces. In computers, as in brains, different stored information can become active at different times. This is related to the point about conditional instructions below. It is also connected with Ryle's emphasis (Ryle 1949) on the *dispositional* properties of minds. Minds, like computers, have many dormant dispositions that will be activated when conditions are appropriate. This fine-grained control of a very large number of distinct capabilities is essential for most biological organisms.

F3. Conditional transitions based on boolean tests:
For many physical systems, behaviour changes continuously through additive influences of

[9] I have produced fragments of the following list of features previously, e.g. chapter 5 of (Sloman 1978), but have never really put them all together properly. The list is still provisional however. I am not aware of any other explicit attempt to assemble such a list, though I believe that all the points are at least intuitively familiar to most practising AI researchers and software engineers.

multiple continuously varying forces. The behaviours of such systems can typically be represented by systems of differential equations. In contrast, there is a small subset of physical systems, including some organisms and some machines, in which behaviours switch between discrete alternatives on the basis of boolean (or more generally discrete-valued) tests. The laws of such a system may include conditional elements, with forms like

"if X then do A else do B"

possibly implemented in chemical, mechanical or electronic devices. Systems controlled by such conditional elements can easily be used to approximate the continuous dynamical systems as is done every day in many computer programs simulating physical systems. However it is hard to make the latter simulate the former — it requires huge numbers of carefully controlled basins of attraction in the phase space. One way to achieve that is to build a machine with lots of local dynamical systems which can be separately controlled: i.e. a computer!

F4. Referential "read" and "write" semantics:

If the behaviour of a system is to be controlled in a fine-grained way by its internal state, the active elements of the system need some way of accessing or interrogating relevant parts of the internal state, for instance in testing conditions or selecting control signals (instructions) to become active. Likewise if a system is to be able to modify its internal state in a controlled way so as to influence future behaviours, it needs to be able to modify parts of itself so that they can be interrogated later.

In principle, there are many ways this ability to interrogate or change specific components can be implemented. In computers it involves allowing bit patterns in memory and in address registers to be interpreted, by the machine, as referring to other parts of the machine: a primitive kind of "referential semantics" — discussed further in (Sloman 1985; Sloman 1987), where it is argued that this can form the basis for many other kinds of semantic capabilities. In early computers the reference was to specific *physical* parts of the system. Later it was found more useful to support reference to *virtual* machine locations, whose physical implementation could vary over time.

F5. Self-modifying laws of behaviour:

In some machines and organisms, the features listed previously can be combined in such a way that the machine's laws of behaviour (i.e. the stored programs) can be changed while the machine is running by changing the internal state (e.g. extending or modifying a stored program). Such changes involve internal behaviours based on features F2, F3 and F4). Such self modification can be useful in many different ways include long term changes of the sort we call learning and short term changes where what is sensed or perceived at a point in time can be stored long enough to be used to modify subsequent actions. I suspect that the full variety of self-modifications in animals, also required for sophisticated robots, has not yet been appreciated, for instance changes which alter the virtual machine architecture of a human during development from infancy to adulthood.

F6. Coupling to environment via physical transducers:

We have implicitly been assuming that some parts of a system can be connected to physical transducers so that both sensors and motors can be causally linked to internal state changes.

If external sensors asynchronously change the contents of memory locations, that allows the above "read" capabilities to be the basis for perceptual processes that control or modify actions. Likewise if some locations are linked through transducers to motors, then "write" instructions changing those locations can cause signals to be transmitted to motors, which is how internal information manipulation often leads to external behaviour. Thus sensors can write information into certain memory locations which can then change subsequent internal behaviour, and some of the memory locations written to by the internal processes can cause signals to be sent to external motors. This implies that the total system has multiple physical parts operating asynchronously and concurrently.[10] Perception and action need not be restricted to "peephole" interactions through very narrow channels (e.g. transmitting or receiving a few bits at a time). Where large numbers of input transducers operate in parallel, it is possible for different perceptual patterns at different levels of abstraction to be detected and interpreted: multi-window perception. Likewise, concurrent multi-window actions can occur at different levels of abstraction.

Interesting new possibilities arise if some of the transducers asynchronously record not states of the external environment but *internal* states and processes, including those in physical and virtual machines. In computers this plays an important role in various kinds of error checking, memory management, detection of access-violations, tracing, and debugging. See also the section on self-monitoring, below.

I believe all of the above features of computers were understood at least intuitively and implemented at least in simple forms by the late 1950s and certainly in the 1960s. None of this depended on knowledge of Turing machines.

We now turn to "secondary features" of computers. These are usually not inherent in the hardware designs, but can be added as virtual machines implemented on the basis of the core features F1 to F6. (In some computers they are given hardware support.)

Some of these features were needed for general programming convenience, e.g. enabling chunks of code to be shared between different parts of the same program, and some were needed because computers were interacting with an unpredictable environment (e.g. a human or some other machine). None of these features was required *only* for AI purposes. In other words they were all part of the continuing general development of the two main historical strands: producing more sophisticated, flexible, and autonomous systems for controlling physical machinery, and producing more sophisticated mechanisms for manipulating abstract structures.

F7. Procedure control stack: the ability to interrupt a process, suspend its state, run some other behaviour, then later resume the original suspended process.
This feature facilitated programs with re-usable modules that could be invoked by other modules to which they would automatically return when complete. Later versions allowed the re-usable modules to be parametrised, i.e. to have different "local" data on different invocations (unlike GOSUB and RETURN in BASIC). This allowed recursive procedures to perform slightly different tasks on each recursive activation. This feature, along with feature F10 below (support for variable-length information structures) was particularly important in supporting some of the descriptive, non-numeric, AI techniques discussed in (Minsky 1963).

Eventually all this was facilitated in electronic computers by hardware or microcode support for a "control stack", i.e. special memory operations for suspended process descriptors, plus mechanisms for pushing and popping such process descriptors. This depends on the saved process states being specifiable by relatively small state descriptors which can be rapidly saved and restored. It would be hard to do that for a neural net, or a mechanical loom. It can be done by mechanical devices, but is far easier to implement in electronic mechanisms.

It might be done by a *collection* of neural nets, each implementing one process, where only one can be in control at a time. This presupposes a fixed process topology, unless there is a supply of spare nets that can be given new functions. The "contention scheduling" architecture (Cooper

[10] In (Sloman 1985; Sloman 1987) I discuss how the system can interpret these sensory changes as due to events in an external environment whose ontology is quite different from the machine's internal ontology.

and Shallice 2000) is something like this.

F8. Interrupt handling:
The ability to suspend and resume processes also allowed a system to respond to external interrupts (new sensory input) without losing track of what it had been doing previously. This kind of asynchronous interrupt is unlike the previous case where control is transferred by an explicit (synchronous) instruction in the program which is to be suspended. Asynchronous interrupts can be supported by software polling in an interpreter. Faster, lower level, support built in to the computer's hardware mechanisms reduces the risk of losing data and sometimes simplifies software design, but the key idea is the same.

F9. Multi-processing:
On the basis of extensions of the previous mechanisms it became fairly easy to implement multi-processing systems which could run many processes in parallel on a single CPU in a time-shared fashion, with interrupts produced at regular intervals by an internal clock, instead of (or in addition to) interrupts generated by external events. This requires larger contexts to be saved and restored, as processes each with their own control stacks are switched in an out.

Such multi-processing allows the development of virtual machine architectures that permit variable numbers of concurrent, persistent, asynchronously interacting processes, some of them sharing memory or sub-routines with others. It provides more flexibility than could be achieved by wiring together a collection of computers each running one process, since then the maximum number of processes would be fixed by the number of computers.

Multi-processing virtual machines allow new kinds of experiments with mental architectures containing variable numbers of interacting virtual machines, including various sorts of perceptual, motivational, learning, reactive, deliberative, planning, plan execution, motor-control, and self-monitoring processes. This is a far cry from the notion of AI as the search for "an algorithm" (as discussed by Searle (Searle 1980) and Penrose (Penrose 1989), and criticised in (Sloman 1992)).

In the early days of AI research focused on algorithms and representations, whereas in the last decade or so, there has been a growing emphasis on complete systems with architectures that include many sub-mechanisms running concurrently. One consequence is that different algorithms can interact in unexpected ways. This is important for the design of a fully functioning intelligent robot (or animal) with multiple sensors, multiple motors, in a changing and partly unpredictable environment, with a variety of more or less independent motive generators operating asynchronously under the influence of both internal processes and external events (Beaudoin 1994).

However, I don't think that many serious experiments of this sort were done by AI researchers before the 1980s. Such experiments were drastically impeded by speed and memory limitations at that time. (Examples were some of the blackboard architectures. My POPEYE system in the late 70s (Sloman 1978) was an attempt at a visual architecture with different concurrent mutually interacting layers of processing, but it was against the spirit of the time, since the influence of Marr was dominant. (Marr 1982))

F10. Larger virtual data chunks:
Another feature of computers which became important both for AI and for other purposes (e.g. databases, graphic design) was the ability to treat arbitrarily large chunks of memory as "units". There were various ways this could be done, including reserving a collection of contiguous (physical or virtual machine) locations as a single "record" or "array" with some explicit information specifying the beginning and the length so that all the memory management

and memory access mechanisms respected that allocation.

More subtle and flexible mechanisms used list-processing techniques implemented in software (though there have been attempts to provide hardware support to speed this up). This allowed the creation of larger structures composed of "chained" pairs of memory locations, each containing some data and the address of the next link in the chain, until a "null" item indicated that the end had been reached. This allowed new links to be spliced into the middle of a chain, or old ones deleted, and also permitted circular chains (of infinite length!).

The full power of this sort of mechanism was first appreciated by McCarthy and others interested in logic and recursive functions. But it is a generally useful technique for many purposes that have nothing to do with AI or cognitive science, and was bound to be re-invented many times.[11]

Thus, while machine hardware usually treats a fixed size bit-pattern as a chunk (e.g. 8, 16, 32, or 64 bits nowadays), software-enabled variable-size virtual structures allow arbitrarily large chunks, and also allow chunks to grow after creation, or to have complex non-linear topologies. The ability of humans to memorise words, phrases, poems, or formulae of varying length may be based on similar capabilities in brains. Similar considerations apply to the ability to perceive and recognize complex objects with different sorts of structures, such as wheels, cars, lorries, railway trains, etc.

As noted in Minsky's 1961 paper (Minsky 1963) various processes involved in perception, use of language, planning, problem solving, and learning, required the ability to create structures that were as large as needed. It also allowed structural units to be complex trees or networks, as opposed to simply being a bit or bit pattern, or a linear array of items.

Support for variable size, variable topology data chunks is not a requirement for ALL kinds of minds. E.g. it seems unlikely that insects need it. Even if bees, for instance, are able to learn routes or topographic maps with variable length and topology, this can be implemented in chained associations, which have many of the properties of list structures mentioned above. (This was discussed in Chapter 8 of Sloman 1978, in connection with how children learn about numbers.).

Variable size and variable structure chunking appear to be required for many animal capabilities, including: learning about terrain, learning about social relationships and acquiring new complex behaviours, e.g. the kinds of problem solving "tricks" that can be learnt by various mammals and birds. It seems to be essential for many aspects of human intelligence, including mathematical problem solving, planning, and linguistic communication (though it has recently become fashionable in some circles to deny this.)

Of course, it is clear that in humans (and other animals) the sizes of the manageable unitary information chunks cannot be *arbitrarily* large, and likewise the data-structures in computers are limited by the addressing mechanism and the physical memory. The infinite tape of a Turing machine is an idealisation intended to overcome this, though typically unusable in practice because of performance limitations of a memory that is not randomly addressable. For an organism with a finite life-span operating in real time, however, there is no need to have space to store unbounded structures.

A single animal brain can use different sorts of chunking strategies for different sub-mechanisms. In particular there are various buffers used for attentive planning or problem solving or reflective thinking, or for real-time processing of sensory data. These all have dedicated mechanisms and limited capacity. So there are limits on the sizes of chunks that can be created and directly manipulated in those short term memories, some of which (e.g. low level visual arrays)

[11]However, the usefulness of general purpose list-processing utilities will usually be severely restricted in programming languages that are strongly statically typed, like most conventional languages.

may be much larger than others.

Similar size limits need not hold for chunks stored in a longer term memory whose chunks may be too large for instantaneous attention (e.g. a memorised poem, or piano sonata, or route, or one's knowledge of a familiar town or building).

F11. Self monitoring and self control

Previously, when discussing sensory transducers in feature F6, above, I mentioned the possibility of computers having sensory transducers asynchronously detecting and checking *internal* states and processes in addition to *external* ones. The need for this sort of thing has been growing as more and more robust, reliable, secure systems have been developed.

To some extent this self-monitoring can be implemented in software, though it is clumsy and slows things down. E.g. it could use the following design:

(i) every instruction (in a class of "monitorable instructions") saves a description of what it has just done in some data-structure.

(ii) a process which examines the saved descriptions is time-shared with the other processes. Note that this process may also save descriptions of what it has done.[12]

It may turn out that one of the most important directions for future developments will be to extend these mechanisms. Requirements of advanced AI systems with various kinds of self-awareness may include hardware (firmware?) architectures that provide some kind of general-purpose self-monitoring capability. This might use one or more additional CPUs running the reflective and meta-management processes required to enable such self-percepts to be analysed, parsed, interpreted, evaluated, quickly enough to be of use in helping to make the whole system more intelligent.[13]

Simply recording the flow of machine instructions and register contents as a collection of state vectors may not be good enough: it is too low level, though I presume that could be done easily with current technology, at a price. It would be better to have direct support for language-specific or VM-specific records, recording larger chunks. I.e. the self-monitoring mechanisms may have to be parametrisable, like the procedure invocation records in a procedure call stack.

Feature F11 is probably implemented in a very different way in neural systems in brains: e.g. as a neuron fires and sends signals to various other neurons as part of doing its job, there will not be much interference with its performance if an additional connection is made to a monitoring network getting information from many other parts of the system. Compare the "Alarm" mechanisms depicted in our recent papers on the CogAff architecture (Sloman 2000a; Sloman and Logan 2000; Sloman 2000b; Sloman 2002).

The features of modern computing systems listed above can all be seen as continuations of the two trends of development that started in previous centuries and were accelerated by the advent of electronic mechanisms that replaced mechanical and hydraulic storage mechanisms, sensors, switches and connections. They do not owe anything to the idea of a Turing machine. But they suffice for all the computer-based work that has been done in AI and cognitive science so far, in addition to all the other types of applications of computers.

Even if computers as we know them do not suffice for future developments it seems unlikely that what will be needed is something like a Turing machine. It is more likely that speed, power and size requirements may have to be met by special purpose hardware to simulate neural nets,

[12] A distributed architecture for self-monitoring using mutual meta-management is outlined in (Kennedy 1999).

[13] For discussions of the role of a meta-management layer in human minds see (Beaudoin 1994; Sloman 1997; Wright *et al.* 1996; Sloman 1999; Sloman 1998; Sloman 2000a; Sloman 2002; Sloman 2001a)

or perhaps chemical computers that perform huge numbers of computations in parallel using interacting molecules, e.g. DNA. Turing machines do not appear offer anything that is missing.

6 Are there missing features?

Some AI researchers may think the list of features of computers presented above leaves out important features needed for AI, e.g. unification, pattern matching, rule-interpreters, support for logical deduction. However I believe these additional features are required only for more specialised AI models and applications. Rather I have tried to identify what the *most general* features of the physical mechanisms and the virtual machines found in computers are that make them important for AI and Cognitive science. This has little or nothing to do with logic, mathematics, or Turing machines, all of which are concerned with rather specialised types of information processing. I shall also try to show, below, that these general features are very similar to features of brain mechanisms, as if evolution discovered the need for these features long before we did.

I have not explicitly included arithmetical capabilities in the 11 main features, though in current computers they are part of the infrastructure for many of the capabilities described above, e.g. calculation of offsets in data-structures, handling relative jumps in instruction counters, scheduling and optimising processes, etc.

A possible objection to the above list of features is that it omits an important human capability, namely the ability to cope with infinity. This ability was noted by Kant (Kant 1781), who tried to explain our grasp of infinite sets of numbers and infinite space and time in terms of our grasp of rules which can be applied indefinitely. Frege's analysis of sentences and their meanings as "compositional" (Frege 1960), pointed to our ability to grasp indefinitely complex grammatical and semantic structures. This was stressed by Chomsky (Chomsky 1965), who claimed that human beings have a kind of *infinite competence* in their ability to understand or generate sentences of unbounded complexity, e.g. arithmetical sentences and sentences like:

the cat sat on the mat
the brown cat sat on the mat,
the big brown cat sat on the mat
the big brown cat sat on the mat in the corner, . . . etc.

The obvious objection is that human life is finite and that the brain is finite, and any individual will produce only a finite number of sentences before dying, and would in any case be unable to understand sentences above a certain length. To this Chomsky replied that even though there are *performance* limitations that get in the way of applying infinite competence, the *competence* exists nevertheless – an answer that left many unconvinced. Questions about the capabilities of computers generate a similar disagreement, as we shall see. One of the important tasks for a theory of mind and computation is to resolve the apparent conflict between infinite competence and bounded physical resources. Both sides are saying something correct, but they talk past each other.

The infinite (or indefinitely extendable) tape of a Turing machine was designed to support this kind of infinite, or unbounded, competence: that was part of Turing's motivation for the unbounded tape, since he was trying to account for human mathematical abilities. Human mathematical abilities appear to be unbounded in various ways. For instance there is no largest number that we can think of – if there were we could think of its square. Likewise there is no largest algebraic expression whose meaning you can grasp. It seems that if there were you could append " + 1" and still understand it, at least with a bit of help, e.g. introduce a new name for the complex expression, such as "BigExp", then think about BigExp with " + 1" appended.

Must this infinite capability be built in to the underlying physical hardware of computers if they are to be adequate to represent human minds? If so, modern computers would fail, since they have finite memories, and although it is often possible to add memory to a computer, it will have bounded addressing capabilities which limit the size of addressable memory. E.g. a machine with only N-bit addressing mechanisms can make use of only 2^N distinct locations.

The situation is more complex than this suggests, for two reasons. One is that additional addressing capabilities can be built on top of the hardware addressing mechanisms, as happens, for instance, when a computer uses a segmented memory, where one address selects a segment and another selects the location in the segment, or when it uses a large backing store, or uses internet addresses to refer to a larger space of possibilities than it can address directly in RAM. However, even those techniques merely *enlarge* the address space: it remains finite.

There is a more subtle point which does not depend on extending the underlying addressing limits. We know that existing computers, despite their finiteness as *physical* mechanisms, can provide implementations for infinite virtual machines, even though the implementations, if fully tested, turn out to be incomplete. That is because they can store, and apply, algorithms that have no bounds.

For instance, in many programming languages it is possible to give a recursive definition of the factorial function which expresses no limit to the size of the input or output numbers. When such a function runs on a computer, either in compiled or interpreted mode, it accurately executes the algorithm for computing the factorial of the given number, up to a point. However, the implementation is usually incomplete in that beyond a certain range of integers the process would abort with some sort of error instead of computing the correct answer.

This does not stop us (correctly) thinking of this as an implementation (albeit only partial) of an infinite virtual machine. That is because every process can be described at different levels of abstraction, and there is a level of abstraction at which the process can be described perfectly correctly as running the recursive algorithm, which does not include any size limits. That is *exactly* what the computer is doing at that level of description, even though *how* it is doing it raises problems if too large a number is given as input.

The point can be expressed by considering which generalisations are true descriptions of the process. For example the virtual machine that is running on the computer has the feature that if it is given the task of computing the result of dividing the factorial of any positive number N, by the factorial of N-1, the result will be N. Similarly, we can say of a sorting algorithm that the result of applying it to any list L will be a new list the same length as L. In both cases the actual physical implementation would break down if the input were too large. If that happened only because there was not enough RAM it might be possible, on some modern computers, to add another bank of memory while the machine was running so as to allow the process to continue, or alternatively it might be possible to kill some other less important processes so as to make more memory available. In that case we can see that the memory limit is a *contingent* or *inessential* feature of the implementation.

We can now see that the fact that a process might run out of memory is analogous to the fact that a stray alpha-particle might corrupt the memory, or the power supply might fail, or the building containing it might be struck by a large meteorite etc. There are vast numbers of possible occurrences that would prevent normal completion of the calculation. We don't feel we have to mention all of those when asking what result the computer would produce if given a certain task. There is a perfectly natural way of thinking about processes running on a computer which treats running out of memory as being analogous to being bombed. If the machine were adequately defended, radar might detect the bomb and divert and destroy it before it hit the building. Likewise additional mechanisms might protect the process from running out of memory.

It is not easy to extend the addressing limits of a machine at run time! However, it is not beyond the bounds of possibility to have a machine that generalises what existing paged virtual memory systems do, namely detect that there is not enough space in the main fast memory and then move some unused pages out and thereafter keep changing the "working set". A more general virtual memory mechanism for a computer that could extend its usable physical store indefinitely would need some addressing scheme that did not use any fixed length structure for specifying locations. In fact, it could, in principle use one or more Turing machines with indefinitely extendable tapes. Whatever method is used, the process of specifying the next location to access and the process of fetching or storing something there might get slower and slower, as happens in a Turing machine.

The point of all this is that when something like the standard recursive factorial algorithm runs on a computer the fact that there is a limit to the size of input it can cope with is an *inessential* feature of the current implementation, which might be changed while the machine is running. Thus we can ask counterfactual questions about what the result would be if the input were the number 2,000,000 while assuming that implementation obstacles are circumvented as they turn up, including such obstacles as the limited size of the physical universe. Of course, if someone intended the question to be interpreted differently, e.g. so that the answer takes account of the current mechanisms in the machine, or the run time modifications that are currently feasible, then this constraint could be explicitly included in the question, and a different answer might then be relevant, e.g. "the computer would run out of memory before finding the result".

In exactly the same sense, human brains, not least the brains of expert mathematicians, can include implementations of infinite machines, albeit partial and buggy implementations insofar as the machines will falter or fail if given inputs that are too complex, or, for that matter, if tiredness, a distraction, alcohol or some other drug interferes with performance.

In short, then, although no human mind can actually do anything infinite because of its current implementation, nevertheless humans (and possibly some other animals?) have unbounded virtual machines as part of their information processing architecture.

We already know that such virtual machines can be implemented usefully in physical machines that do not support the full theoretical range of functions like factorial which they partially implement. Thus neither human minds nor robots with human-like intelligence need have something with the infinite capacity of a Turing machine in the physical implementation in order to be running a virtual machine with unbounded competence.

What is more, insofar as the human architecture includes a meta-management layer that can (to some extent) inspect the operations of the system, it is able to discover such facts about itself. This is the source of much of the philosophy of mathematics which is concerned with, among other things, the ability of humans to think about infinite sets. Likewise a robot whose mind is implemented on a computer and includes a meta-management layer could discover that it had a kind of infinite competence, and might then become puzzled about how that could be implemented in a finite brain.

All this leaves open the question what precisely is going on when we think about infinite sets, e.g. the set of positive integers, or the transfinite ordinal consisting of the set of all powers of two followed by the set of all powers of three, followed by the all powers of five, and so on for all prime numbers. I have discussed this inconclusively in (Sloman 2001b) and will not pursue the matter here.

7 Some implications

The important features of computers which I have listed (as we know them now, and as they may develop in the near future) have nothing to do with logic, recursive functions, or Turing machines, though systems with the features that are important in computers can (subject to memory limits) model Turing machines and can also be modelled by Turing machines, albeit excessively slowly (assuming that part of a requirement for a functioning mind is to react in time to seize opportunities and avoid dangers).

Turing machines and modern computers can both also model neural nets, chemical soups using DNA molecules to perform computations, and other types of information processing engines,

Given all that, the view of computers as somehow essentially a form of Turing machine, or as essentially concerned with logic, or recursive functions, is simply mistaken. As indicated above, there is such a mathematical notion of computation, which is the subject of much theoretical computer science, but it is not the primary motivation for the construction or use of computers, nor is it particularly helpful in understanding how computers work or how to use them. A physically implemented turing machine (with a finite tape) would simply be a special case of the general class of computers discussed here, though linking it to sensory or motor transducers might cause problems.

Neither is the idea of a Turing machine relevant to most of the capabilities of human and animal minds, although, as explained above, there is a loose analogy between a Turing machine and some of the formal capabilities of human minds, including the ability to think and reason about infinite structures. This must have evolved relatively late and does not play an important role in the vast majority of everyday mental processes. It does not appear to be relevant to most animals or to very young children if, as seems likely, these abstract reasoning abilities develop during childhood rather than being there from birth.

8 Two counterfactual historical conjectures

The preceding remarks are closely connected with two conjectures:

1. The development of computers was an inevitable consequence of availability of new electronic technology providing the potential to overcome the limitations of previously existing machines for controlling other machines, and machines for managing and analysing information (e.g. commercial databases, airline reservation systems, etc.). Consider what happened during the World War II: the need to decipher encrypted messages accelerated development of electronic calculators.

Most of this would have happened anyway, even if Turing and others had not done their meta-mathematical work on computation, computability, derivability, completeness, etc.

2. If Turing had never existed, or had never thought of Turing machines, and if computers with the sorts of features listed above had been developed under the pressure of requirements for increasingly sophisticated (and fast) control of physical machines and processing of information for practical purposes, then much of AI and Cognitive Science would have developed exactly as we know them now. AI would not miss the concept of a Turing machine.

Some people interested in logic, theorem proving, etc. might have noticed that such computers could be used to implement logic machines of various kinds. But there would have been no notion that computers were somehow *inherently* logical, or *inherently* related to mathematical theorems and their proofs.

Those two conjectures follow from the argument that computers were, and are, above all, engines for acquiring, storing, analysing, transforming, and using information, partly to control physical machines and partly in order to control their own processing of information. In this they are just like biological organisms — and that includes using the information both to control complex physical and chemical systems, and also to control internal processing within the controller, including processing in virtual machines.

If we think of computers like this, then it is an empirical question whether a particular physical object does or does not have the kinds of capabilities listed above. It is not true that every physical object has the collection of features F1 to F11, or even the simpler set F1 to F6. However, finding out which features a very complex machine actually has can be a very difficult reverse-engineering problem. The best available tools at a particular time (e.g. brain scanners) may not be adequate for the job.

ACKNOWLEDGEMENTS

This research is partly funded by a grant from the Leverhulme Trust.

I have profited from discussions with Margaret Boden and Matthias Scheutz.

I strongly recommend Minsky's 'Steps towards Artificial Intelligence' (first published in 1961), as an excellent overview of ideas about AI that had developed by about 1960. It remains an extremely important paper, although at that stage neither he nor others had appreciated the importance of the study of architectures for combining multiple concurrently active capabilities – as opposed to the study of representations and algorithms. It is interesting that the 1961 paper anticipated some of the mathematical results on perceptrons later published by Minsky and Papert, although he also stresses that neural nets can be useful as part of a larger system. This was written 20 years before the rise of connectionism, and even longer before the fashion for investigating hybrid architectures.

The Birmingham Cogaff project includes many papers exploring requirements for intelligent systems and gradually evolving specification for the CogAff architecture. See
`http://www.cs.bham.ac.uk/research/cogaff/`

Gary Forbis kindly pointed out a typo in the published version of this paper.

References

[Beaudoin 1994] L.P. Beaudoin. *Goal processing in autonomous agents*. PhD thesis, School of Computer Science, The University of Birmingham, 1994. (Available at http://www.cs.bham.ac.uk/ research/ cogaff/).

[Chomsky 1965] N. Chomsky. *Aspects of the theory of syntax*. MIT Press,, Cambridge, Mass, 1965.

[Cooper and Shallice 2000] R. Cooper and T. Shallice. Contention scheduling and the control of routine activities. *Cognitive Neuropsychology*, 17(4):297–338, 2000.

[Frege 1960] Gottlob Frege. In P. Geach and M. Black, editors, *Translations from the Philosophical Writings*. Blackwell, Oxford, 1960.

[Haugeland 1985] J. Haugeland. *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, MA, 1985.

[Hodges 1983] Andrew Hodges. *Alan Turing: the Enigma*. Burnett Books, London, 1983.

[Kant 1781] I. Kant. *Critique of Pure Reason*. Macmillan, London, 1781. Translated (1929) by Norman Kemp Smith.

[Kennedy 1999] C. M. Kennedy. Distributed reflective architectures for adjustable autonomy. In *International Joint Conference on Artificial Intelligence (IJCAI99), Workshop on Adjustable Autonomy*, Stockholm, Sweden, July 1999.

[Marr 1982] D. Marr. *Vision*. Freeman, 1982.

[Minsky 1963] M. L. Minsky. Steps towards artificial intelligence. In E.A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 406–450. McGraw-Hill, New York, 1963.

[Pain 2000] Stephanie Pain. Liquid assets. *New Scientist*, (2268):46–47, 2000.

[Penrose 1989] Roger Penrose. *The Emperor's New Mind: Concerning Computers Minds and the Laws of Physics*. Oxford University Press, Oxford, 1989.

[Ryle 1949] G. Ryle. *The Concept of Mind*. Hutchinson, London, 1949.

[Searle 1980] J.R. Searle. Minds brains and programs. *The Behavioral and Brain Sciences*, 3(3), 1980. (With commentaries and reply by Searle).

[Sloman and Logan 2000] A. Sloman and B.S. Logan. Evolvable architectures for human-like minds. In G. Hatano, N. Okada, and H. Tanabe, editors, *Affective Minds*, pages 169–181. Elsevier, Amsterdam, 2000.

[Sloman 1978] A. Sloman. *The Computer Revolution in Philosophy*. Harvester Press (and Humanities Press), Hassocks, Sussex, 1978. Online at http://www.cs.bham.ac.uk/research/cogaff/crp.

[Sloman 1985] A. Sloman. What enables a machine to understand? In *Proc 9th IJCAI*, pages 995–1001, Los Angeles, 1985.

[Sloman 1987] A. Sloman. Reference without causal links. In J.B.H. du Boulay, D.Hogg, and L.Steels, editors, *Advances in Artificial Intelligence - II*, pages 369–381. North Holland, Dordrecht, 1987.

[Sloman 1992] A. Sloman. The emperor's real mind. *Artificial Intelligence*, 56:355–396, 1992. Review of Roger Penrose's *The Emperor's new Mind: Concerning Computers Minds and the Laws of Physics*.

[Sloman 1997] A. Sloman. What sort of control system is able to have a personality. In R. Trappl and P. Petta, editors, *Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents*, pages 166–208. Springer (Lecture Notes in AI), Berlin, 1997.

[Sloman 1998] A. Sloman. Damasio, Descartes, alarms and meta-management. In *Proceedings International Conference on Systems, Man, and Cybernetics (SMC98), San Diego*, pages 2652–7. IEEE, 1998.

[Sloman 1999] A. Sloman. What sort of architecture is required for a human-like agent? In Michael Wooldridge and Anand Rao, editors, *Foundations of Rational Agency*, pages 35–52. Kluwer Academic, Dordrecht, 1999.

[Sloman 2000a] A. Sloman. Architectural requirements for human-like agents both natural and artificial. (what sorts of machines can love?). In K. Dautenhahn, editor, *Human Cognition And Social Agent Technology*, Advances in Consciousness Research, pages 163–195. John Benjamins, Amsterdam, 2000.

[Sloman 2000b] A. Sloman. Interacting trajectories in design space and niche space: A philosopher speculates about evolution. In M.Schoenauer et al., editor, *Parallel Problem Solving from Nature – PPSN VI*, Lecture Notes in Computer Science, No 1917, pages 3–16, Berlin, 2000. Springer-Verlag.

[Sloman 2001a] A. Sloman. Beyond shallow models of emotion. *Cognitive Processing: International Quarterly of Cognitive Science*, 2(1):177–198, 2001.

[Sloman 2001b] A. Sloman. Diagrams in the mind. In M. Anderson, B. Meyer, and P. Olivier, editors, *Diagrammatic Representation and Reasoning*. Springer-Verlag, Berlin, 2001.

[Sloman 2002] A. Sloman. How many separately evolved emotional beasties live within us? In R. Trappl, P. Petta, and S. Payr, editors, *Emotions in Humans and Artifacts*, pages 29–96. MIT Press, Cambridge, MA, 2002.

[Turing 1950] A.M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950. (reprinted in E.A. Feigenbaum and J. Feldman (eds) *Computers and Thought* McGraw-Hill, New York, 1963, 11–35).

[Wright *et al.* 1996] I.P. Wright, A. Sloman, and L.P. Beaudoin. Towards a design-based analysis of emotional episodes. *Philosophy Psychiatry and Psychology*, 3(2):101–126, 1996. Repr. in R.L.Chrisley (Ed.), *Artificial Intelligence: Critical Concepts in Cognitive Science*, Vol IV, Routledge, London, 2000.