

# Can Computers Be Genuinely Intelligent?

*Peter Kugel*

*Computer Science Department, Boston College*

*(Abstract 132 words, text 16,587 words.)*

## **Abstract**

The conventional wisdom has it that Turing (1950) argued that computing machines can become genuinely intelligent. I don't believe it.

I think that what he really tried to say was that computing machines – computers limited to computing – can only *fake* intelligence. And that, before computers can become *genuinely* intelligent, they will have to be given enough "initiative" (Turing 1969) so that they can do more than compute.

In this paper, I want to develop this idea. I want to explain how giving computers "initiative" can make it possible for them to do more than compute, and why I think they will have to do that before they can become genuinely intelligent.

I also want to look at how this idea might affect the way we deal with intelligence in machines and people.

## **1. Computing Machinery and Intelligence**

People who try to make computers intelligent say they are trying to create Artificial Intelligence (or AI). Presumably, they use the word "artificial" to suggest that the intelligence they are trying to produce will – like artificial vanilla – not have developed naturally.

But some of their critics believe that anything that looks like computer intelligence will have to be artificial in another sense – the sense in which an "artificial smile" is artificial. Which is to say faked. Computers, they believe, cannot be genuinely intelligent because they lack an essential ingredient that genuine intelligence requires.

The more passionate of these critics believe that what computers lack is so fundamental that it cannot be given to them. Some suggest that intelligence requires an immortal soul. Others feel that it can only be found in systems made of flesh and blood. Such critics believe that computers cannot be genuinely intelligent, period.

Other critics of AI are a bit more generous. They believe that, although computers don't have it all yet, they can be given what they need for genuine intelligence. If you give computers access to neural nets, quantum computers, analog devices or some other kind of extra system, they will (with our help, of course) be able to achieve genuine intelligence.

In this paper, I want to suggest that the two sides of this debate are both right and both wrong. I believe that computers already have everything that they need to become genuinely intelligent, but that no amount of work along the lines that most of AI is going today will get them there. Today's AI is riding the right horse, but it's taking it down the wrong road. I want to suggest that the problem lies, not in our computers, but in ourselves. We are not using computers the way they will have to be used if we want them to become genuinely intelligent.

If you step back from the details, it is not hard to see where the problem lies. Turing (1969, but written in 1947) saw it more than fifty years ago when he suggested that, in order to be intelligent, computers would have to be given what he called "initiative." That should not be surprising. You have to let people exercise initiative if you want them to behave intelligently. You cannot expect intelligence from people if you insist that they do exactly what you tell them to do in exactly the way you tell them to do it.

Yet, as long as we use computers to compute, we are doing precisely that. And that's where the problem lies. As long as we continue to assume that computers will be able to compute their way to intelligence we will (I want to argue) fail. Computers can

do more than compute and, I want to suggest, that is precisely what they will have to do if we want them to behave really intelligently.

Computing is, of course, what computers were invented to do, but that does not mean that they cannot be used to do other things. Screwdrivers were invented to drive screws, but that does not mean that they cannot be used to pry open paint cans.

One of the things it means for a computer to compute is that it must give us its results with a sense of finality. If you ask a computer to multiply 1,234 by 567, you expect it to give you the answer (699,678) and stop. You want a result that you can take back to your laboratory or accounting department.

You don't want the computer to act up. You don't want it to add "...but I'm not sure." And you certainly don't want it to call you back later – perhaps after you have used the number it gave you to approve a drug, or report your company's earnings – to say it has “changed its mind.” If anybody is going to change their mind in a transaction between you and your computer, you want it to be you. (“Sorry, I made a typing mistake. I wanted you to multiply 1,234 by 667.”)

And that, I claim, is the problem. If you give a computer enough "initiative" to allow it to change its “mind” it can do things that it cannot do if you limit it to computing. And one of those things is to behave intelligently.

Turing seems to have realized this. Speaking to the London Mathematical Society in 1947, he said (Turing, 1986):

...if a machine is expected to be infallible, it cannot also be intelligent. There are several mathematical theorems which say almost exactly that. But these theorems say nothing about how much intelligence may be displayed if a machine makes no pretence at infallibility.

Turing did not live long enough to develop this idea very far, but he seems to have given it some thought. He seems to have realized that saying that computers must do more than compute before they can become genuinely intelligent does not tell you much. It is a bit like saying that China is not in South America. It tells you where computer intelligence is not, but not where it is.

If you believe that it takes more than computing to become intelligent, you need a map of the uncomputable to say where intelligence is. Turing tried to develop such a map in his Ph.D. thesis (Turing, 1939), but he did so in terms of what he called "oracles" that he described in terms of infinite tapes. And the trouble with infinite tapes is that they cannot be physically realized.

What he managed to show was that the land of the uncomputable has a geography – that some functions are more uncomputable than others. But he did not tell us where in this geography intelligence lies nor how we might implement machines that could take us there. (Copeland and Proudfoot, 1999)

Some time after Turing's death, Putnam (1965) and Gold (1965), independently, developed accounts of certain levels of uncomputability that lend themselves more readily to machine implementation. And it is among these kinds of uncomputability that, I want to suggest, the extra something that intelligence requires might be found.

## **2. Computing and Thinking**

The ideas behind this suggestion are a bit subtle, and to make them clear, we will have to look quite carefully at the ideas of computability and uncomputability. If you

define "computable" carelessly – in a way that seems quite natural – you end up being able to prove that the general-purpose computer that most of us have on our desks cannot exist. Turing (1936) was aware of this (although, of course, he did not have a computer on his desk). He came up with a definition of "computable" that avoids this problem, and that we now accept, when he was working on a mathematical problem known as the *Entscheidungsproblem*. This problem asks: "Can we develop a single algorithm (or what we would now call a 'computer program') that will determine, for any given set of axioms (A) and potential theorem (T) whether or not T can be proved from A?"

Hilbert, who proposed the *Entscheidungsproblem*, and gave it its hard-to-pronounce-in-English name, believed that such an algorithm could be found. Since proving theorems is an important part of what mathematicians do, a procedure based on such an algorithm would make it possible to automate this part of their work. (If, for example, such an algorithm had existed, it could have been used to compute the answer to the question: "Can Fermat's last theorem (T) be derived from the axioms of arithmetic (A)?" which would have saved mathematicians a great deal of trouble.)

Turing disagreed with Hilbert. He felt that no such general-purpose algorithm was possible. But it is one thing to feel that something is impossible, and quite another to prove it mathematically. To do that, Turing needed precise definitions of the two basic concepts involved – the idea of a *computation* and the idea of a *proof*.

He developed both and showed that, if you accept them, no general-purpose algorithm for determining provability was possible. This was a significant accomplishment. But the definition of *computation* that he developed in the course of

generating his proof, and a theorem he proved about it, turned out to be even more important than his negative solution to the *Entscheidungsproblem* because they played a central role in the development of the digital computer.

His definition gave us a precise characterization of what is computable in principle. His theorem told us that it is possible to build a single machine that can do all of those things. Once you had such a machine, you had only to tell it which among all the possible computations you wanted it to carry out (by giving it what we would now call a “program”) and it would carry it out for you.

Turing developed his characterization of computability by analyzing what the human mind does when it computes. He found that he could describe that in terms of a few rather simple operations. And he defined a class of abstract machines that could perform these operations. Today, we call these machines *Turing machines* and it is generally agreed that what these machines can do precisely captures the intuitive idea of what it means to be computable. (Notice, by the way, the irony in this. The issue in this paper is whether such machines, based on an analysis of the mind, are a suitable model for the mind. My answer will be “No” and the questions this paper asks are “Why not?” and “What is missing?”)

Turing gave a precise definition of what it means for a theorem,  $T$ , to be provable from a set of axioms,  $A$  by showing how a computer could go about generating such a proof (if one existed). But although a computer can generate a proof, if one exists, what it cannot do is decide whether a proof of a potential theorem,  $T$ , exists. If a proof exists, its theorem grinder-outer will generate it. But if none does, a computation will not

always be able to determine that. And that is why no computation can solve the *Entscheidungsproblem*.

It is important to notice that computers can prove theorems. What they cannot do is to do it as well as the *Entscheidungsproblem* demands. (Theorem proving can be done by a partial computation, but a solution to the *Entscheidungsproblem* demands a total computation.)

In order to prove his theorem, Turing developed the idea of a single universal computing machine – one machine that can do everything that any computing machine can do if it is told how to do it. This abstract machine is the ancestor of today's programmable computer and Turing subsequently developed it in three directions:

- In the *practical direction*, he helped implement a physical universal machine in the form of ACE, one of the world's first electronic digital computers. (Turing, 1986a)
- In the *theoretical direction*, he started to explore the theory of what his universal computing machines could not do. (Turing, 1939)
- In the *scientific direction*, he suggested how those machines might be used to model human cognition. (Turing, 1950)

Turing's suggestion that the computer might be a good model for the human mind appealed to many students of that mind, and it plays a crucial role in today's cognitive science. At first glance, the idea that a machine as simple and mindless as the computer could be made intelligent seems implausible (especially if you have seen how monumentally stupid they can sometimes be). But the fact that computations are, themselves, unintelligent does not necessarily mean that they cannot combine to

produce intelligence. After all, neither hydrogen nor oxygen is wet, but that does not mean that they cannot combine to produce water, which is. That is one of the things that scientific theories try to do. They try to show how complex phenomena can be built up from simpler elements, thereby helping us to understand them.

Turing saw the promise of this possibility and, some fifteen years after he wrote his paper about the *Entscheidungsproblem*, he wrote a paper (Turing, 1950) in which he asked whether computing machines could be programmed to be intelligent. In that paper, he is widely believed to have argued that they could. I want to suggest that what he really argued was that they could not.

What he tried to say, I believe, was this:

- It will probably be possible to program computing machines – limited to carrying out computations – so that they can *fake* intelligence well enough to fool human beings. (To some extent, that prediction has come true.)

- It will probably not (repeat not) be possible to program such machines – limited to computations – to be *really* intelligent. (I believe that this prediction has come true too.)

Turing's paper is better known for its first conclusion (although the business about faking it is usually played down). His second conclusion is often ignored, perhaps because Turing said so little about it. What he did say, he said near the end of his 1950 paper, where he wrote:

Intelligent behaviour presumably consists in *a departure from the completely disciplined behaviour involved in computation*, but a rather slight one, which does not give rise to random behaviour, or to pointless repetitive loops. (my italics)

In this paper, I want to suggest one kind of “departure from the completely disciplined behaviour involved in computation” – a “rather slight one” – that might make genuine computer intelligence possible. And I want to suggest that this departure may be important, not just for the study of computer intelligence, but also for study of the natural kind, which you and I like to believe we have in abundance.

### **3. Computers and Cognitive Science**

The role that computations play in the cognitive sciences today is very much like the role that numbers play in the physical sciences. Just as people working in the physical sciences often ask that accounts of the physical world be expressed in numerical terms, so people working in the cognitive sciences today generally ask that accounts of cognitive processes be expressed in computational terms. The belief of many cognitive scientists is roughly this: "If you can't give a computational account of a cognitive process, you haven't really explained it."

This assumption helps keep cognitive scientists "honest" in several ways. One thing it does is to prevent them from ignoring critical details. When you try to give a computational account of a process – say the process by which we recognize occurrences of the letter “A” – you have to fill in details that you might otherwise ignore.

Another thing it does is that it helps prevent them from adopting theories that could not work. Once you fill in the details of a computational account you can program it for a computer, run the program, and see whether it does what you think it does, before you look to see whether the way the program does it is the same as the way we

do it. And, if your program really does what you think it is going to do, you get a third benefit. A computer, running your program, can do it for you.

It is largely because they enjoy these benefits that cognitive scientists are so eager to protect the view that "thinking can be reduced to computing" against its critics. They are worried that, if this claim turns out to be unfounded, their science will again be reduced to kind of arm-waving that it used to be when it was based on introspection, or the kinds of relatively trivial results it was able to obtain when it was based on outwardly observable behavior alone.

I understand this motivation, and I would like to suggest that, even if genuine intelligence is beyond the reach of computability, it may not be necessary to discard computers as models for intelligence. It may be possible to extend the idea of the computer as a model in such a way that you end up keeping most of its benefits while, at the same time, stretching it enough to include what intelligence requires.

There is a striking historical precedent for this hope. The physical sciences did something very much along these lines when they ran into a similar problem with the rational numbers. When mathematicians discovered things that rational numbers could not handle, they were able to extend those numbers to include new ones – what today we call the "irrational" real numbers (like  $\pi$  and  $\sqrt{2}$ ). These new numbers covered the problematical cases, but they also did something else. They provided new conceptual tools that made a more powerful mathematics possible. And that mathematics ultimately made a more powerful physics possible. (Ironically, the irrationality of  $\sqrt{2}$ ,

which the Pythagoreans feared might "ruin" the hopes for a science of the physical world, actually made a better science of that world possible.)

I want to suggest that a similar move might make sense in the cognitive sciences. Perhaps we can extend the computing-machine model for cognition to allow some *uncomputing-machine* models that will extend the set of conceptual tools available for explaining the mind. These extended tools might then let us keep the benefits of computer models of the mind, while at the same time making it possible to deal with the more difficult cases. And, in the process, they may give the cognitive sciences a more powerful mathematics.

Let me remind you how the story of the physical sciences and the numbers developed because the story of the cognitive sciences and the computers could be quite similar.

Once upon a time, the Pythagoreans believed that the physical world could be fully characterized numerically. To their minds, it came down to a simple two-way choice. Either the physical world could be fully characterized in terms of numbers, or it could not be characterized scientifically at all. And, by "numbers", they meant "rational numbers" – the kind whose values can be completely expressed as a ratio between two integers like  $2/1$ ,  $1234/567$ , or  $-7/3$ .

They set about studying such numbers, and they discovered some interesting things about them. For example, they discovered what we now call "the Pythagorean Theorem" according to which the length of the diagonal of the unit square (the square whose side is one unit long) is  $\sqrt{2}$ . And they discovered that  $\sqrt{2}$  could not be expressed

by a rational number. They discovered, in effect, something in the world that looked as though it could not be characterized by a number.

At first, this upset the Pythagoreans, and it is said that they tried to keep the irrationality of  $\sqrt{2}$  a secret to protect their claim that “all is number”. They might, instead, have argued for a science of “Artificial Numbers” (or “AN”). Pythagoras might have suggested an “imitation game” to show that  $\sqrt{2}$  was really a rational number. Put the real  $\sqrt{2}$  in one room, and a rational approximation – say to 100 decimal places – in another. Let a human judge construct a unit square and measure its diagonal. Let the judge then ask the two rooms for the “correct” value of  $\sqrt{2}$  to whatever number of decimal places he or she has determined. If the rational  $\sqrt{2}$  can give the required number of decimal places, then it wins the game and shows that there is no need to add an irrational  $\sqrt{2}$  because the artificial one is indistinguishable from a real one for practical purposes. (In 1897, the Indiana State Legislature considered – but did not pass – a bill legislating the equivalence of  $\pi$  to a rational approximation on the grounds that it was sufficiently accurate for all practical purposes.)

AN is a useful enterprise, and we still indulge in it. But western science also did something else. It kept its faith in the real  $\sqrt{2}$  and simply expanded the concept of number to include it, along with the other irrational numbers.

I want to suggest that a similar move might be appropriate in the cognitive sciences. Let us keep the computing machines as our basic models for cognitive processes. But let us admit that there are certain aspects of intelligence that computations cannot fully characterize.

Let us expand the idea of what computing machinery can do in roughly the way we expanded the rational numbers to include the irrationals. We know that there is no decimal fraction that will exactly express  $\sqrt{2}$ . But we also know how to get closer and closer to  $\sqrt{2}$  by computing longer and longer decimal approximations. 1.41 is close. 1.414 is closer, 1.4143 is closer yet, and so forth.  $\sqrt{2}$  is the number you would get by carrying out this process (each of whose steps can be calculated) forever.

Mathematicians today define such numbers, without using the word "forever", by talking about "passing to the limit". Gold (1965) and others have suggested that something very much like this can be done with the idea of a computation, and that allowing it extends what computers can do. It allows them to do what I propose to call *uncomputations*.

Since the basic ideas behind such an extension are a bit a bit hard to understand, there are three things that you might ask me to promise you before you go through the trouble of trying to work your way through the rest of this paper:

- That I will spell out what it means to extend computing machines in this way – what it means to compute something "in the limit."

- That I will show you some feature of intelligence that cannot be characterized by ordinary computations, but can be characterized in terms of such limiting computations, much as the diagonal of the unit square can be characterized by  $\sqrt{2}$ , but not by a rational number.

- That I develop the theory of such "limiting computations" in a way that is as brilliant as Newton's (and Leibnitz's) development of the calculus, and show you how to

use this expanded theory to develop a cognitive science as impressive as Newton's physics.

Well, I hope you'll be satisfied with two out of three. A full-fledged, Newton-level cognitive science, based on a mathematics of computing in the limit that is as rich as the calculus, remains a hope for the future and I can give you only a vague indication of how it might go. (For some more specific indications, see Martin and Osherson (1998) and Jain, Osherson, Royer and Sharma (1999).)

#### **4. Why Computations Are Not Enough**

To see what's wrong with the idea that the mind can only compute, imagine what it would be like if it that were all it could do. Imagine that humans are born with a mind that has the abilities of a general purpose computing machine and that it has a few programs, or what corresponds to programs, in it to start. (We might call these built-in programs *instincts*.) Assume furthermore that experience somehow develops new programs for this machine as they are needed. (We might call the process by which this comes about *learning*.) It is not hard to see that there is something drastically wrong with this idea. To see what it is, we let's take a closer look at some of Turing's ideas and fill in a few details.

Turing based his precise definition of a computation on the kinds of computations we do in grammar school mathematics – computations that, like addition and multiplication, always produce results. Computations that have this property are said (by mathematicians) to be *total*.

Computations can also be *partial* in the sense that they may produce no results for some (or even all) inputs. Division, for example, produces no result when the divisor is 0. But we usually know when this will happen. There are partial computations for which we cannot effectively determine when they will fail to produce a result. They may just keep running without producing any output and your problem, as their user, is that you don't know whether a result will be forthcoming if you wait long enough or not.

When Turing tried to define his "universal" machine, he discovered that he had to make a choice. He could avoid those situations in which a computation would give no results for some inputs. But he found that, if he did that, he could prove that the resulting machine was not universal – that there would be some totally computable functions that it could not compute.

On the other hand, he found that he could make the machine universal, but only if he allowed some of its programs to be partial. He had to choose, and he opted to include the partial functions so that the machine could be universal.

If you want to install a programmable computing system into a human brain, you face the same choice, and that is where your problem lies. You can install a universal machine, in which case it can, in principle, develop a program to handle anything that can be handled computably. But then it could come up with partially computable functions and, when it tried to use one of those functions on a particular input, it might not produce any results. On the other hand, you could install a limited machine that can only learn to handle totally computable functions. If you do that, you will be forced to leave out some (totally) computable functions that your limited machine cannot compute.

If we call a machine that can compute any totally computable function *universal*, and a machine that computes only total functions *total*, then your dilemma is this. If you want to endow a human being with the capabilities of a programmable computing machine, the machine can be either total or universal, but not both. And your problem lies in the fact that both possibilities could be fatal to the infant whose design you are in charge of.

Today's general-purpose computer is universal, but not total. We rely on programmers to make sure that the no-result cases don't arise. That allows us to buy a single machine for our desk, knowing that it can be programmed for any purpose we might want – if it can be handled by a computation at all.

But that could be a risky choice for a human mind. To see why, let's try to design an infant. Our goal is to provide it with the mental machinery that will allow it do the computing it needs to survive – to find food, avoid predators, stay near its mother while it is young, and so forth. Let's pick one of these tasks, say avoiding predators.

The easiest thing to do – and something that nature has almost certainly done in some cases – is to install a special-purpose machine with a single program (or finite set of programs) that allows the organism to deal with a limited set of predators. But if we want our infant to be able to deal with environments that its genetic makeup has not anticipated – and human beings seem to be rather good at this sort of thing – we will have to make its machinery a bit more powerful.

So we might endow its machinery with some “knobs” that it could use to adjust parameters for its predator-avoiding programs. That could allow our infant to learn to deal with a broader set of predators, but it would still be limited. We could do even

better if we gave the infant, at birth, the machinery of a general-purpose computer that its experiences could (somehow) program. And this is where our choice comes in. We have to decide whether to endow it with a total machine (which evaluates only totally computable functions) or a universal one (which evaluates all totally computable functions, but some partially computable ones too).

At first glance, a total machine might seem to be our best bet. If we chose such a machine, the infant would never find itself facing a predator for which its current program gave no result. It is not hard to see the benefits of this. It would be less likely to become the lunch of a predator for which its current program was unable to compute a response.

But, if we choose a total machine to avoid this possibility, our infant might face a different problem. It might find itself facing predators for which its total (but not universal) machine could find no program. It would face a predator that it could not "understand", and it might end up as that predator's lunch.

It is true that, for certain kinds of learning situations, a total machine makes sense. For example, in his analysis of the machinery the child might use to learn the grammar of its native language, Chomsky (1957) proposed a version of this alternative, and current evidence suggests that he was probably right. But such a move seems more plausible for those situations in which the same evolutionary path evolves both the producer and the learner of a set of behaviors – as it does in the evolution of the speakers and the learners of human languages. It is less plausible when the predator producing the behavior, and the human dealing with it, evolve through different and competitive paths.

Fortunately there is a third alternative that offers a plausible compromise.

Suppose we let the mind be universal. Then, to handle the cases in which the selected program might produce no output, let the mind's computer start by putting forth (mentally) an automatic response to use in case this is one of those no-output cases. If it needs a result before the current computation gives one, let it use the automatic response. If we do that, we can get universality – as well as faster answers when we need them – while avoiding the disasters that a no-response-is-computed case might produce.

But, and here's the kicker, if we do that, we end up with a machine that – although it seems an awful lot like a computing machine – can evaluate uncomputable functions. What such a machine does – plugging any possible holes in its computations before they begin – doesn't seem like much of a change, but it is. (I warned you that we would have to be careful about definitions.) As soon as we allow such hole-filling, our infant can do things that no computing machine can do.

(From here on in, I will use the phrases *computing machine* and *computer* to refer to a computer limited to carrying out computations and the terms *uncomputing machine* and *uncomputer* to refer to machines that use the same physical machinery to do more than compute)

Uncomputing machines can, for example, solve the *halting problem* which no computing machine can solve (Turing, 1936). The halting problem is the problem of determining whether an arbitrary computing program, run on an arbitrary input, will or will not halt. To see that a hole-filling machine can solve it, we use a fact on which the universality of the universal machine depends – that we can program a universal

computing machine to simulate the program of any other computing machine. Given a program, Prog, and an input, Inp, such a simulation imitates the behavior of Prog(Inp) (the program, Prog, running on the input, Inp) and outputs a YES if the Prog(Inp) halts and a NO if it does not.

With this procedure in hand to use as a subroutine, it is not hard to write a program for our gap-filling machine that will solve the (computationally unsolvable) halting problem. Given Prog and Inp, let our machine begin by outputting NO (to indicate that Prog(Inp) will not halt). Then let it run the simulation of Prog(Inp) and, if the simulation halts, let our program output YES. It is easy to see this program's last response will always correctly tell us whether Prog(Inp) will or will not halt.

There are two ways of looking at the differences between such procedures and computations. One is that you can compute how long computations take but not (in all cases) how long uncomputations take. (Computations take computably long to come up with their results. Uncomputations can take uncomputably long.) The other way of looking at the difference is that a computation is allowed at most one chance to produce an output for a given input whereas our gap-filling procedure is allowed (at most, but not at least) two. Following Putnam(1965), let us call an ordinary computation a *one-trial* procedure and this gap-filling kind a *two-trial* procedure.

From the machine's point of view, a two-trial procedure is allowed to make a guess and then "change its mind". From a user's point of view, such a procedure produces results that may or may not be final. (In the case of a two-trial procedure, only the first output can be changed. The second, if it comes up, is final.)

Such two-trial machines are only the first of a series of uncomputing machines that can be implemented by an ordinary computer. There is an infinite series of such machines, each more powerful than the preceding one. A three-trial machine (allowed at most three tries) is provably more powerful than a two-trial kind and, in general, an  $(n+1)$ -trial machine is more powerful than an  $n$ -trial machine. (Kugel, 1977.)

If you allow an unlimited (but at most finite) number of outputs, or trials, for each input and count the *last* output a machine produces as its result, saying that the result is undefined if either there is no first or no last, you get an even more powerful kind of machine that we might, following Putnam (1965), call a *trial-and-error machine*.

You cannot distinguish such uncomputing machines from normal computing machines by looking at how they are made or how they operate. Both kinds follow the instructions of their programs a step at a time in a strictly "mechanical" way. What's different is how we interpret what they do. With a computing machine, we take its *first* output as its result. With a trial-and-error machine, we take its *last* output as its result. Unless such a machine shuts itself off, all of its outputs are tentative results because subsequent operations can always change them. (Such machines make "no pretense at infallibility", to quote Turing's (1986) words.)

Notice how the results produced by trial-and-error machines are defined "in the limit" in very much the way the values of irrational numbers are defined. (Gold, 1965) Just as it takes infinitely long to write down the decimal expansion of  $\sqrt{2}$ , so it can take infinitely long to determine the (final) result of a trial-and-error procedure. That can be something of an inconvenience, so it would be nice if we could get intelligence without using such procedures. If Turing (1986) was right, we might not be able to.

To show that this kind of "fallibility" is necessary for intelligence, we need something in the cognitive world that corresponds to the diagonal of the unit square in the physical world – something that intelligent systems can do that cannot be characterized by a computation, much as the diagonal of the unit square cannot be characterized (precisely) by a rational number. What we need, in other words, are aspects of intelligence that can only be implemented by uncomputations. And here we face a mildly annoying problem. When we talk of intelligence, we don't really know what we are talking about. There seems to be no generally accepted definition of "intelligence."

Let's go look for one.

## **5. Uncomputability and Machine Intelligence**

With the growth of public education toward the end of the 19<sup>th</sup> century, the French Ministry of Public Education noticed that a few children were disrupting the classroom because they seemed unable to learn. It asked Alfred Binet to develop a test to identify them. The test Binet developed is now called an "intelligence test". Its aim was to measure the ability to learn and so, to start us off, let's assume that intelligence is (at least in part) the ability to learn.

To learn what? We learn lots of different kinds of things, but many of them can be represented by computer programs. And many of the things we learn, we learn from examples. So let's go further and assume, not only that intelligence is the ability to learn, but also that it is the ability to learn, from examples, what can be represented by computer programs. That seems to be what the child does when it learns its native

language from the utterances it hears and what it does when it learns the concept of "dog" (or "doggie") after seeing a few tail-wagging examples.

Finding a program is quite different from running one. When we run a program we go from a fully specified function,  $f_n$ , to its values  $f_n(1)$ ,  $f_n(2)$ , and so forth. When a child learns the functions that characterize (say) some feature of its native language, the process it uses seems to go "the other way" – from the values  $f_n(1)$ ,  $f_n(2)$ , and so forth (or, in the case of language, specific utterances) to the function,  $f_n$ .

When you go from the values to the function, you "go beyond the information given", to use Bruner's (1973) felicitous phrase. You go from a finite set of values given by experience to the infinite set that your function determines. And when you do that, some of the values you "predict" may be wrong even if all the values you are given are correct. If you know that  $f_n(1)$ ,  $f_n(2)$ ,  $f_n(3)$ , and  $f_n(4)$  are all 2, there is nothing that prevents  $f_n(5)$  from being 73. When you hear that the plural of "house" is "houses" and of "grouse" is "grouses", that does not prevent the plural of "mouse" from being "mice".

This process of going from the values of the function to a program for evaluating that function is, I believe, a good model for the process of going from evidence to theories. Is it also a good model for at least part of what is involved in intelligence? I think so.

Notice that Binet's tests tried to measure this ability indirectly. Most IQ tests do not try to see how well you can learn, but how well you have already learned. And they do this by seeing what "programs" you have now learned to use. There are obvious practical reasons for this, and it is how we tend to determine the intelligence of the people we meet. We see what they know and can do. But I suspect that what we often

have in mind when we think of intelligence is not the knowledge we have acquired but the ability we have to acquire it.

Some questions on IQ tests seem to try to measure this ability somewhat more directly. One popular type of question on such tests asks you to extrapolate sequences of integers. You might, for example, be asked to continue the sequence **2,4,6,8**... The "correct" continuation is, of course, 2,4,6,8,**10,12,14,16**,... but that is not the only possibility. The sequence might have been 2,4,6,8,**2,4,6,8,2,4**.... or 2,4,6,8,**8,8,8,8**... or even 2,4,6,8, **who, do, we, appreciate**. The first continuation may be the "right" one, but a person who stuck to it as a "theory" after being told that the fifth item in the sequence was 2 or 8 or "who" would be a fool. Yet that is exactly what a computing (or one-trial) machine would have to do if it chose its theory after seeing the first four (or any finite number of) elements.

That does not seem very smart, and that is one reason why I believe that computations are not a good model for intelligence. After they come up with their results, they are forced – by the definition of "computation" – to be stupid. They are forced to stick with their results.

Now consider the machine that is allowed to keep the equivalent of an "open mind". Let it proceed by trial and error. When it has seen enough evidence to think it has a theory, let it output that theory. But let it also change that theory when it no longer fits the evidence. And let's count, not the first theory it comes up with as its result, but the last.

The behavior of a trial-and-error machine is exactly like that of a computing machine and you can't tell them apart by looking at them. What distinguishes them is

that we interpret their behavior differently. We can shut off a computing machine when it gives us its first result because we know that that is the result that counts. But we cannot do that to a trial-and-error machine because, if we shut it off after its first output, we will not know its last and that is what counts.

The trial-and-error machines are one kind of uncomputing machine. There are many others, but the trial-and-error machines are the paradigmatic ones. And they are just the kind of machine we need to avoid the kind of inconsistency (or "stupidity") that the limitation to computing seems to force on a system trying to develop a theory from examples.

Notice, by the way, that avoiding this kind of "stupidity" does not guarantee intelligence. The quality of the theories a system comes up with matters too. The theory that classifies emeralds as "grue" (green up to now but blue ever-after) may remain consistent with the evidence without being particularly "intelligent". (Goodman, 1954) Being a trial-and-error machine may not be sufficient for intelligence but, I would like to suggest, it is probably necessary.

There are other kinds of questions on IQ tests that seem to call for the use of "mind changing" (uncomputing) machines. Consider, for example, questions that ask the person taking the test to categorize – to distinguish squares from circles, birds from dogs, and the like. The categories have to be learned of course, but even after they have been learned, they have enough baroque edges and open-ended-ness to suggest the use of uncomputations. As Wittgenstein (1958) has pointed out, the structure of the concept of "game" seems to behave like an extendable conglomerate into which new items can always be inserted. The way we extend concepts over time, or narrow them

with experience, shares some of the flexibility of the way we develop theories and may involve some of the same kinds of uncomputable procedures.

The *Entscheidungsproblem* is about categorization. It asks for an algorithm that can determine whether or not a particular statement, T, falls under the concept "provable from a set of axioms A". When Turing proved that the *Entscheidungsproblem* was unsolvable he proved that the concept of provability was not computable.

But the set defined by this concept can be characterized by an uncomputation. Consider a process that generates all the possible proofs of theorems from a given set of axioms. It is easy to show that, under quite natural assumptions, this can be done by a computation, and we say that any set whose members can be listed completely by a computing program is *recursively enumerable*. Turing showed that this set – representing the extension of a concept – was not computable. But recursive enumerability (or partial computability) is so close to total computing that few people consider it to be really uncomputable. (It is not computable because, although the problem of determining whether a statement, T, is in the set of theorems provable from the axioms, A, can be solved by a computation, the problem of showing that T is not provable cannot.)

The set of all truths of elementary number theory is also uncomputable. It is not recursively enumerable (which is what Gödel's (1931) famous incompleteness theorem says), but it too is still within the range of a realizable uncomputing machine.

Consider for example, a version of Gödel's theorem due to Rosser (1936), according to which no formal system of arithmetic is capable of proving a theorem that

expresses its own formal consistency. (A system is said to be *formally consistent* if it is impossible to prove two theorems of the form  $T$  and  $\neg T$ , where  $T$  is a well-formed-formula and " $\neg$ " represents negation.)

It is not hard to devise an uncomputing procedure to determine the formal consistency of any formal system of arithmetic. Since the theorems of any such system can be generated (recursively enumerated) by a computing machine, consider using the program that implements such a procedure as a subroutine. Start your off your procedure for ascertaining the consistency of a formal system by having it assert the consistency of the system (by outputting YES) and then letting it generate all the theorems of that formal system one at a time. Each time it generates one, let it check the new theorem against all the theorems already generated for formal consistency. If the procedure finds an inconsistency, let it "change its mind" and assert that the system is inconsistent by outputting NO. It is easy to see that, if we count the last output that this procedure produces as its result, it correctly determines the formal consistency of the axiomatic system in question. Such a machine is a two-trial machine.

Two-trial machines can also solve the *Entscheidungsproblem*. Given the problem: "Is the theorem  $T$  provable in the axiomatic system  $A$ ?" we simply start our machine off by having it "guess" NO and then letting it grind out all the theorems of  $A$ . If it finds  $T$  on its list, it changes its mind and outputs YES. Voila!

If human beings have the power of two-trial machines, these observations provide a mathematical basis for two claims sometimes made about the human mind:

- That it can "violate" Goedel's theorem, and therefore is not a computing machine, a claim made most famously by Lucas (1961).

- That it can "solve" the *Entscheidungsproblem*, and therefore is not a computing machine.

It is easy to show that two-trial procedures can do everything that one-trial (or computing) procedures can do and more. And, of course, a trial-and-error machine can do anything a two-trial machine can do. What is more, we also know that, for any positive integer  $n$ ,  $(n+1)$ -trial machines are more powerful than  $n$ -trial machines and that trial-and-error machines are more powerful than  $n$ -trial machines for any fixed  $n$ . What is more,  $n$ -trial machines that start with a NO can solve problems that those that start with a YES cannot and vice versa. (The mathematical arguments behind these observations, and others yet to come, are sketched in an earlier paper of mine (Kugel, 1986).)

We also know that trial-and-error machines are not complete when it comes to theory-finding. No single trial-and-error machine will come up with the right theory for any possible (computably enumerable) sequence from the elements of that sequence. And we know how to construct a theoretical machine that can do a complete job of this. It is more powerful than any trial-and-error machine, and such more powerful machines might provide still other models for what is involved in genuine intelligence.

The true statements of elementary arithmetic form what has been called a "productive set" (Dekker, 1955). A set  $S$  is said to be *productive* if it is not recursively enumerable and there is a productive function,  $C$ , such that, if somebody claims to come up with a program,  $P$ , for recursively enumerating  $S$ ,  $C$  generates a program  $C_P$  that enumerates an infinite subset of  $S$ , none of whose members are in the set enumerated by  $P$ .

Myhill(1952) suggested that concepts in art, such as what an art counts as beautiful, might form productive sets. If, for example, we think of the people who teach music appreciation as teaching a theory of what counts as "beautiful" in music and we assume that such a theory recursively enumerates all the pieces of music that should be so counted, then Myhill suggests that any such proposal can be extended to produce new sets of "beautiful" music that were not included in the original account. (See also Kugel, 1990.) That, by Myhill's account, is what a composer, like Stravinsky, does when he develops a new style that goes beyond the traditional canon of his day. It is, in other words, because we are uncomputers that art has the expandable structure it has.

Another place where "trial-and-error" procedures seem to show up is in the process by which we go about understanding the meanings of words. Thus, for example, the word "fly" has several meanings in English. If somebody asks me "How do you make an elephant fly?" my mind computably finds its way into one of these. But it doesn't settle in there because, when that question is followed by the punch-line: "Start with a 48-inch zipper," my mind drops the first meaning and substitutes another.

This sort of thing – changing what we understand the meaning of a word to be as we see or hear more text – seems to be quite common, and is not limited to understanding (bad) jokes. We commonly change the way we interpret the meanings of the words we read as we read more. Compare "He hit the fly..." followed by "with a fly-swatter" and followed by "with a baseball bat."

The meanings of words in specific utterances seem to be flexible, and such flexibility seems to play a crucial role in our ability to understand some kinds of humor

and some aspects of literature. The way we assign such meanings as we listen to or read text seems, to me, to involve uncomputations.

There are other aspects of human cognition that might be modeled by uncomputing machines. Notice, for example, that human beings (outside of psychology labs) seldom deal with such limited questions as "Is this an instance of the letter A?" They are more likely to consider such questions as "What letter is this?" or perhaps even more often "What is this?"

Here is one possible account of how the mind might use uncomputations to go about recognizing what letter a particular mark on paper represents. Imagine that the mind has separate processors for each of the 26 letters of our alphabet. When it sees a squiggle on paper that it wants to recognize, it sets all 26 of these processors going at once, with each one trying to *disprove* that the squiggle represents the letter it is assigned to recognize. When it comes time to decide what letter the squiggle represents, the mind picks the least-disproved one, but the others continue to go about their business.

Unlike Selfridge's (1959) Pandemonium model, which chooses from among halting processes, this process chooses from among non-halting ones which is why it involves uncomputability. Such families of non-halting machines offer another source of models for cognitive processes in both machines and people. And it suggest some possible designs for parallel uncomputing machines, some of which take us up the hierarchy of uncomputable processes beyond the trial-and-error machines.

The ability to solve problems is often thought of as a component of intelligence. McCarthy (1956) has suggested that certain kinds of problem solving might be modeled

by a process of "inverting computable functions". When we compute a function,  $f$ , we take the function, together with an argument,  $x$ , and compute the value,  $f(x)$ . When we invert a computable function, we go the other way. We are given the function  $f$  and a value  $v$ . Our job is to find an  $x$  such that  $f(x)=v$ . One way of doing this is to try all the possibilities,  $f(1)$ ,  $f(2)$ ,... until we get to one that evaluates to  $v$ .

That involves uncomputability because there may be no solution. Things get a bit more complicated if we look, not for some particular value,  $v$ , but for the "best" value by some computable measure. Now if we are willing to accept a high enough level of "badness", some solution can always be found, but we can never stop looking for a better unless we find one that is "perfect". There are various strategies for solving this kind of problem, and different strategies may define different sorts of intelligence. One thing that is interesting about these differences is that there is no clearcut way of saying that one is "better" than another. (Kugel, 1986)

Measures of intelligence – the so-called *intelligence tests* – tend to measure intelligence on a single linear scale and come up with a single number. If we knew more about the structure of intelligence, we might be able to develop more powerful, multi-dimensional, tests of intelligence that would come up with more complex intelligence profiles. Such profiles could help us figure out where an individual's intelligence was weak and where it was strong. Knowing that, we might be able to strengthen (or at least work around) the weak areas and take better advantage of that individual's strengths. A theory of intelligence based on uncomputations offers us some possibilities along these lines.

## 6. And so?

Suppose that what I have been saying is true.

Suppose that there really are two kinds of intelligence – fake and genuine.

Suppose that you can fake intelligence with computations alone and that, if you want to be genuinely intelligent, you have to do more than compute. And suppose that computers, in spite of what we call them in English, can do the kinds of "uncomputations" that genuine intelligence requires. What would it mean?

One thing it would not mean is that we ought to get rid of fake intelligence.

Although my choice of the names "genuine" and "fake" suggests that the former is better than the latter, that is not necessarily the case. Fake intelligence can be more useful than the genuine kind. It carries out the procedures (applies the theories, uses the concepts, exercises the skills) that genuine intelligence derives from experience. Fake intelligence harvests the fruits of real intelligence without growing them. It has all the benefits of a supermarket over an orchard.

When implemented on a computer, it takes advantage of the intelligence of human programmers, who develop algorithms that the computer then executes computably, to produce results that, when humans produce them, seem to require genuine intelligence. In human beings, fake intelligence uses the products of other people's intelligence to produce results that a single human being could not produce alone.

But things are a bit more complicated in human intelligence because the human mind is a collection of systems that all work together and it is hard to turn off the ones that use genuine intelligence to isolate the fake. So, in human beings, even fake intelligence involves a certain amount of the real stuff.

When teachers tell their students exactly how to do something, what they are doing is quite different from what programmers do when they tell a computer exactly how to do something. The instructions teachers give are not as clear and unambiguous as instructions given in a programming language. Unlike programming languages, ordinary languages are ambiguous and the students have to figure out what their teachers mean in much the same way that people, listening to a joke, have to figure out what is meant by "make an elephant fly".

Fake intelligence in humans is seldom, if ever, pure. Still, the intelligence involved in figuring out Newton's laws in the first place is somehow "more" genuine than the intelligence required to figure it out from being told even if both are genuine.

With humans, the genuineness of intelligence seems to be a matter of degree. A bit of genuine intelligence almost has to creep in, not only in learning ideas by being told, but also after the ideas have been acquired. People naturally use a certain amount of genuine intelligence to adapt their ideas – no matter how mindlessly they may have been learned – in ways that computers usually do not.

But even if the difference between genuine and fake intelligence is less clear in humans, there seems to be one. What is more, it seems to make a difference. People seem to be able to adapt ideas they have learned on their own better than ideas they have learned by being told. A good theory of education might clarify, not only the differences between levels of "genuineness", but also try to explain why ideas one learns on one's own tend to end up more flexible than ideas one has learned by being told.

To get a better feel for what a theory of genuine and fake intelligence in humans might look like, consider the problem of trying to teach cooking. One of the things it means to teach cooking is to teach people how to make a certain set of dishes, defined by recipes. It is easy to see that teaching a person how to follow recipes is not the same thing as teaching computers how to follow programs. When you tell a computer to add two numbers, it does not have to "home in" on addition, nor does it have to "home in" on what its programmer means. But when you teach a person a the skills involved in following a recipe, they have to do both. They do not always understand right away what the instructions mean and even when they do, they do not always immediately know how to carry them out. You typically have to show them what it means to baste a turkey or beat egg whites and then let them try doing it so that they can correct their initial efforts until they get it right. The ability to hone one's cooking skills on the basis of such instructions is one kind of cooking intelligence (or "CI"). It is the kind of low-level genuine intelligence that seems to be required for even the kind of fake intelligence involved in simply following recipes.

There is a second kind of CI that allows cooks to adjust their recipes when things change. Today, the eggs we have in the refrigerator are smaller, the temperature of the room we are letting our bread rise in is higher, or what have you. That means we have to add a little less flour or shorten the amount of time we allow our bread to rise. Such adjustments are little more than changes in parameters and are close to fake intelligence. But they are still a step above.

Then there is a third kind of CI – the kind that allows creative cooks to come up with new recipes. If you were interested in developing this kind of CI, you might begin

by teaching your students how to cook from the recipes of others. That ability would help them put dinner on the table, and it could give them the elements out of which they might then be able to construct new recipes.

As long as what they are learning is fully determined, which is to say as long as they are simply following recipes, we might still call it "fake" intelligence. It gets to be a bit more genuine if you can change the recipes you have learned to produce a different kind of dish – perhaps changing a recipe for chocolate cake into one for banana cake by substituting mashed bananas for the chocolate. The slightly more flexible cooking intelligence involved here seems, somehow, a bit more "genuine". So that's another kind of CI. Your students are still only changing a parameter. (In cooking, it's called an "ingredient".) But it's a more important parameter and they are changing the product in a significant – almost discontinuous – way. But the recipe really is not totally new.

If you wanted to demonstrate an even more powerful kind of CI – the kind that produces a totally new recipe – to a friend, you might be tempted to prepare a dish invented by a famous chef. If you did that, you should not be surprised if your friend pointed out that you were not demonstrating all-out genuine CI because you were using the results of somebody else's CI rather than your own.

People working in AI are often criticized in this way. The kinds of programs that most of today's AI produces seem to follow the "recipes" provided by their programmers who thereby serve as "oracles" (in the sense of Turing (1939)) for their programs. It is, in other words, the programmer and not the program, that demonstrates the intelligence.

(Notice, by the way, how hard it is to determine the kind of intelligence involved by just looking at the behavior. The same cake can be a simple implementation of a recipe from a cookbook, a clever adaptation of such a recipe, or a brand new invention. To see what kind of intelligence is involved, we need to look at where the idea came from and where it goes. We have to look, in other words, at the behavior over time – to study movies rather than snapshots, if you will. This problem is particularly difficult in schools where learning is usually judged by the snapshots we call "tests".)

There are good reasons for wanting to develop genuine intelligence, but there are good reasons for wanting the fake kind too. I suspect that it will pay to try to develop both. And that, I believe, is what Turing (1950), in his paper on "Computing Machinery and Intelligence", may have been suggesting. Contrary to what is widely believed (and widely taught) it is possible to read Turing's paper as suggesting that programming a computer to play the Imitation Game (in which the computer tries to fool a person communicating with it via a computer terminal into believing that it is a person) would be only a first step toward the development of computer intelligence. It would develop the false kind.

The claim that the Imitation Game is not a test for genuine intelligence is supported by the way Turing introduced it. He presented it as a variation of a game in which a man tries to pretend, in a similar situation, that he is a woman. Surely nobody would think that a man becomes a woman by winning such a game. A man in drag is not, after all, a real woman, no matter how clever the deception might be.

In what might be thought of as an earlier draft of his 1951 paper, Turing (1969, but written in 1947) suggested that genuine intelligence might require loosening up the kind

of "discipline" that computation requires. He suggested that it might call for giving computers what he called "initiative." And then he continued:

A very typical sort of problem requiring some sort of initiative consists of those of the form 'Find a number  $n$  such that ....' This form covers a very great variety of problems. For instance problems of the form 'See if you can find a way of calculating the function which will enable us to find the values for arguments....' are reducible to this form.

Finding such a number is, he wrote, "clearly equivalent to finding a program."

(Recall that, for a Turing machine, a program is a number.) And he went on to suggest one way to go about it:

The crudest way of dealing with such a problem is to take the integers in order and to test each one to see whether it has the required property, and to go on until one is found which has it.

Turing understood that such a procedure would not work in practice. For one thing, it would get stuck in trying out a program that never produced a result. (That particular problem can be overcome by a mathematical stratagem known as "dovetailing".) For another, it would have to look through too many numbers to be practical. So he suggested that one way to make it work would be to look for theories in other, more efficient, ways. In this, I believe Turing was right. The development of good algorithms for developing algorithms seems, to me, to be an important step toward the development of genuinely intelligent computer programs.

Algorithms to develop algorithms from evidence, and to adjust existing algorithms to new evidence, should produce algorithms that get closer and closer (or "converge") the correct algorithm. But it is not clear what such convergence involves when what is being changed is a computer program. Nor is it clear how one might bring such convergence about when it involved more than just parameter changes.

It is not, for example, easy to change programs written in today's programming languages in ways that change their behavior in predictable ways. Change one instruction here or there, and the effect on the resulting program can be hard to predict. The development of programming languages in which the description of the program is more closely related to what the program does might be a step toward the development of genuine computer intelligence.

One way to adapt a program to new evidence is to add something at either end of the code. If, for example, a program produces a value that is too small, you just add a piece at the end that increases the size of its output a bit. Or, if it is driven by parameters, you just adjust those parameters. That is surely one kind of adaptation.

But if your theory says that the earth is the center of the universe and it is not working well, it is not quite clear (except of course in retrospect) how one should change it so that it better fits Tycho Brahe's observations. Surely one doesn't change it by moving the earth a little bit. The study of program-changing programs is one direction that research in genuine computer intelligence might take.

At the moment, it is not easy to make a sharp distinction between what one does by changing a program parameter (which produces more or less continuous changes that produce different versions of the same theory) and more basic program changes (which produce changes that seem more "jerky" and produce new theories). This distinction seems clear intuitively, but it is not easy to make precise. Trying to make this distinction more precise is another problem that future research might pursue.

The distinction between genuine and fake intelligence, even if we only understand it rather vaguely, casts an interesting light on what has come to be known as the *Turing*

*Test for machine intelligence.* (A machine is assumed to pass this test, and to qualify as intelligent, if it wins the Imitation Game.) What makes the Imitation Game such a curious test for intelligence is that it seems to focus on programs that do not change much beyond, perhaps, remembering some data about the conversation as it proceeds. Recall that, in that game, the interrogator who is trying to determine which of two terminals is connected to a person and which to a machine, seems to be focusing (as intelligence tests often do) on what the subjects have learned. It is as though an interrogator were trying to test for cooking intelligence by asking questions like "How much celery did you use in your veal stew?" Such questions would focus on what a cook knew, not how well the cook could learn new things or change what he or she already knew.

To get at more genuine intelligence, you might say such things as "We're out of veal. Now what can we make with the other ingredients of veal stew?" and Turing does not seem to have suggested interactions of this kind. He seems to have been focusing on whether the computer's algorithms worked, not how they might be changed.

Thinking about such interactions can help clear up some puzzling things about Turing's test. Consider, for example, the curious problems that arise from a famous thought experiment suggested by Searle (1980) He asks us to imagine a room with a slot in the door. Through that door, someone who understands Chinese can pass in messages written in Chinese. The room responds by passing out messages, also written in Chinese, that, to the person passing in the messages, seem indistinguishable from those that would be passed out by a speaker of Chinese.

Searle asks us to imagine that, inside that room, sits Searle himself -- who understands no Chinese. He is equipped with a script (in English) that tells him how to respond to Chinese writing (represented by what to him are meaningless squiggles) with other Chinese writing (equally meaningless squiggles). And he asks us to imagine that, by following this script, he is able to make responses that are good enough to fool the Chinese-speaking "interrogator" outside the room, into thinking that there is a Chinese-speaker inside.

This room is Searle's analog of the computer playing the Imitation Game and he asks "Where in this room is the understanding of Chinese?"

Well, of course, there is no understanding of Chinese behavior in the room and that is not hard to show if we look for genuine (i.e. flexible) understanding. Suppose, for example, we send in a (Chinese) message that changes the (Chinese) language a bit. The exchange might go like this (translated into English for your convenience and mine. And note that I do not know enough Chinese to know whether it could be expressed in Chinese in the first place):

"From here on in I'm going to use the word 'bad' to mean 'good' as it does in some contemporary American slang. Got it?"

"Yes."

"Would you say that an A was a bad grade?"

"No." (Gotcha!)

The problem arises because the script that Searle is using cannot (if I understand him correctly) be changed by any Chinese experience. The room follows a script that it

cannot change. The Chinese room is an example of the "intelligence is the ability to run a program" view which is not my view at all. To change it into a room that can be said to genuinely understand Chinese, by my account, you would have to give Searle (in the room) the ability to change the script based on the messages he received. You would, in other words, have to make it possible for Searle to understand Chinese.

Could we provide Searle with a script that could, among other things, tell him how to change the script itself? Of course we could, and if that script were well enough designed, the room might well be able to understand Chinese.

One thing such a script might do, for example, would be to develop a model of the world (and of the Chinese language in it) in terms of which it would represent any changes that were made by what it was told (in Chinese). Endowed with such a script, it might even become genuinely intelligent.

But, you might say, such an understanding script is only a script. Well, yes it is, but I never claimed that a scripted room could not be intelligent. I just suggested that Searle's kind of script would not do. Intelligence cannot be intelligence all the way down. At some point it has to be decomposable into components that are not themselves intelligent. And scripts are fine by me.

(There is an old story about William James, who had just given a lecture in which he claimed that the earth floated in space. An old lady came up to him, wagged her umbrella in his face, and said "You're wrong Mr. James. The world rests on a turtle." "And what does the turtle stand on?" asked James. "Another turtle," she replied. "And that turtle? What does it rest on?" The little old lady smiled. "You can't fool me, Mr.

James," she said. "It's turtles all the way down." Well, Mr. Searle, it can't be semantics all the way down either.)

There are reasons to wonder whether developing genuine intelligence in computers might not sometimes be a bad idea. Suppose, for example, we have developed a program to do medical diagnoses, using a trial-and-error procedure. It is not hard to show (Kugel, 1986) that any such procedure is incomplete in the sense that it cannot not make all possible diagnoses. Suppose that the genuinely intelligent diagnosis program – which comes up with its own diagnosis-making procedure – comes up with a procedure that has a completely different scope from the ones that people come up with. Then it could come up with diagnoses that a person could not come up with and therefore could not understand. It is not clear that we would be willing to be treated on the basis of a diagnosis that a person could not understand.

There are at least two ways to avoid this problem. One is to try to simulate human intelligence – the way human beings *learn to make* diagnoses. The other is to try to simulate the way human beings *make* diagnoses. Both approaches might make our programs come to their conclusions in ways that we can understand, and to make mistakes of the kind that we could understand. But the second approach – which is the approach of that part of AI that does cognitive simulation – is probably a good deal easier. Producing programs that make diagnoses, rather than programs that produce programs that make diagnoses may be faking intelligence, but that does not make it a bad idea. It is a relatively easy way to make programs that people can understand.

Helping us understand the program is one reason for making intelligent programs behave like us. There are others. If, for example, you want to program a computer to

play the Imitation Game well, you might want to make your program imitate the shortcomings of human intelligence to fool the interrogator. For example, in Turing's (1950) sample Imitation Game, the following dialog ensues:

Q: Add 34957 to 70764

A: (Pause about 30 seconds and then give as answer) 105621.

Here, to win the game, the computer tries to do a worse job of information processing than it could. It gives the wrong answer slowly, although it could give the right answer quickly. Here it is doing it to fool the judge. But there could be other reasons to do it. To give us programs we can understand is one. To help us understand the way we think is another.

Notice, for example, that the error the computer makes in Turing's dialog is not a random substitution of integers. It is the kind of mistake a person would make if he or she forgot to carry. It is both a mistake that might fool the judge in the game and a mistake that we would understand. And, if we understand such mistakes, we might be able to help people (children learning to add, for example) avoid them.

The distinction between genuine and fake intelligence is hardly new. It is familiar to psychologists. Raymond B. Catell's (1963) distinction between *fluid* (genuine) and *crystallized* (fake) intelligence is a lot like it and similar distinctions can be found widely in the psychological literature.

It also seems familiar to brain scientists. Eichenbaum and Bodkin (2000) have made the interesting suggestion that the products of genuine and fake intelligence – what we call "memories" – might be represented differently in the brain. The results of fake intelligence might be represented in a way that is relatively fixed and inflexible.

Memories that are represented in this way can be tuned, but they cannot easily be transferred or applied to domains outside of those in which they were learned, and they cannot easily be changed by thinking about them consciously.

The results of genuine intelligence, on the other hand, may be represented differently – in a way that allows them to be both transferred more easily to new domains, and operated on by the conscious mind.

The memories acquired by genuine intelligence, Eichenbaum and Bodkin suggest, represent knowledge because they can be more readily adjusted to experience and changed by experience. They may be represented by a network of associations built up, over time.

Eichenbaum and Bodkin suggest that we might distinguish what philosophers call *knowledge* from what they call *belief*, not in the usual terms ("Knowledge is justified true belief.") but in terms of how they are represented in the brain. Beliefs are represented in a way that is relatively fixed and difficult to change. Knowledge is represented in a way that is more flexible and more easily changed as new information comes in. The fact that the latter is more likely to be true and justified is a consequence of that.

To the outside observer, knowledge and belief seem quite similar. Both dispose those who possess them to act in certain ways. The belief that broccoli is good for you, and the knowledge that it is, both dispose you to eat broccoli. Both knowledge and belief are tunable, but they are tunable in different ways. We tune our beliefs by changing parameters. We tune our knowledge in more complex, less continuous, ways.

The way you tune your beliefs is the way you tune your English when you speak louder to somebody who doesn't seem to understand your English. The way you tune your knowledge is more like the way you "tune" your English when you translate what you are trying to say into Spanish after you discover that the person you are speaking to doesn't understand English.

The idea that Eichenbaum and Bodkin add to the distinction between what we might now call *knowledge intelligence* (aka genuine intelligence) and *belief intelligence* (aka fake intelligence) is that these two types of intelligence may differ, not only in how they operate, but also in how their results are represented in the brain. They may, in other words, not only be different trees; they may also produce different fruits.

The full story is probably a good deal more complicated than that. There may be other kinds of intelligence that produce still other kinds of fruits. The human mind, for example, seems to have a kind of intelligence that produces what we might call *feelings* about what we know or believe. These feelings often are also learned from experience. You may, for example, feel differently about riding a horse after you have fallen off one.

The clearest evidence for distinct kinds of memory systems (and presumably therefore distinct kinds of intelligence associated with them) in the human brain, comes from what brain scientists call *dissociation*. Two kinds of brain function are dissociated when one can appear in a person (usually a patient) without the other. Thus knowledge is dissociated from feeling in patients with Capgras' syndrome. (Ellis and Young, 1990) People with this curious brain disorder recognize their parents – they *know* who they are. But seeing them does not give rise to an associated feeling that they really are their parents. So, when they see their mother, they will tell you that they recognize her.

It looks exactly like her but they know it's really only a clever fake. (One such patient is said to have thought that his father was a robot pretending to be his father and killed him to remove the computer chip from his brain.)

It is as though knowledge takes one path through the brain, and feeling takes another. Either of these pathways can be broken. In patients with Capgras's syndrome, the belief pathway is broken. In people with prosopagnosia (Damasio, Damasio, VanHoesen, 1982), in contrast, the knowledge pathway seems to be broken. People with this problem do not recognize their parents. However, their galvanic skin responses show that, when they see their parents, they get the feeling of recognizing their parents, without being aware of it.

Closely related to beliefs are *habits*. Habits are things you can do without thinking about them (such as reading which, until I just drew your attention to it, you were doing with thinking about it). When you drive, you believe that the road will continue to be there in front of you without thinking about it and you exercise the habits of driving without thinking about them either. Habits are dissociated from knowledge in the famous case of HM, a large part of whose hippocampus was surgically removed, leaving him profoundly amnesiac. (Scoville and Milner, 1957) As a result, he could not acquire new knowledge. But he could learn new habits. And there is some evidence that he could acquire new beliefs.

The different kinds of intelligence, and the memories they produce, are distinguished by what can be done to alter them over time. Feelings are probably the hardest to change. If you fall off a horse there seems to be only a short window of opportunity when you can still prevent the more permanent fear of horses that might

result. (You should get right back up. If you wait too long, the feeling that horses are dangerous will be much harder to get rid of.) Beliefs and habits are tuneable, but harder to change in basic ways. You can, for example, adjust your golf swing to the characteristics of a new set of clubs. But feelings, habits and beliefs are not as flexible and transferable as what we have called "knowledge". Their tuneability is one way to distinguish them, but the way they are represented in the brain may be a better way. That's another subject for future research.

The interplay between these different kinds of intelligence – and there are probably many more – seems to be crucial to natural intelligence. We know, for example, that patients who lose access to their feelings – as in the case of a patient Damasio (1995) calls "Elliot" – also lose their ability to reason.

It is interesting to ask whether these distinctions, and others found in the human mind, are accidents of the way intelligence is implemented in the human brain or whether they are distinctions that machine intelligence will have to duplicate. I suspect that computer intelligence may have to duplicate at least some of them but, again, that is something future research might look into.

If, for example, you look at the way people develop theories of numerical sequences such as 2,4,6,8,... you might notice that their theory generation may be flexible, but their arithmetic machinery – which they use to add and subtract, for example – is not. Their flexible genuine intelligence may be built on the now inflexible results of their fake intelligence. These two types may work together, as they appear to when we generate theories of numerical sequences, and both may be necessary to effective cognition.

In spite of the fact that knowledge may be more complex than habit, there are many reasons to prefer habit to knowledge. One is that tuning knowledge seems to require your conscious attention, whereas tuning habit does not. You can carry on a conversation while you are driving a car because your driving is a habit that you can tune to the road, and the traffic on it, without paying attention to either. That lets you focus your attention on other things.

If you hadn't turned your reading ability into a parameter-driven habit that you do not have pay attention to while you are doing it, you would not be able to think about what you are reading right now because your attention would be fully absorbed by the mechanics of reading. You would think about what you are reading a l e t t e r a t a t i m e.

These observations suggest that intelligent thinking, in human beings, may be more like a symphony requiring a collection of instruments, than a solo performance by a single one. The idea that there is more than one kind of intelligence has been proposed by many, perhaps most famously by Gardner (1993). But Gardner cuts the cake of intelligence into slices whereas the more Freudian proposal I have sketched here cuts it into layers.

Efforts to develop computer intelligence might profit in various ways by looking at how intelligence is implemented in the human brain. And, of course, those studying the human brain might profit by looking at what AI learns. The information flow, in other words, can go in both directions.

OK Suppose you agree that there is a distinction between genuine and fake intelligence and, perhaps, a few more distinctions between other kinds. What would you do that was different?

Well, if you were working in AI, it might change the kinds of programs you wrote. You might try to develop uncomputing programs, along with ones that only compute. That could change, not only what AI does, but also how it does it. Today, AI tries mainly to develop programs that solve problems by searching through state spaces, or networks, each node of which represents a static snapshot of a step toward a solution. Theories that try to generate programs to fit data might be thought of as searching through *program spaces* in which the nodes are programs and the arcs, connecting the nodes are.... I don't quite know what they might be, but that is one of the questions an AI, trying to develop genuine intelligence, might ask.

For computer scientists generally, uncomputers offer new challenges. Uncomputers are a special breed of what most of the other contributors to this issue call *hypercomputers* and they have one striking advantage over most of those other kinds. They have already been built. To "get" one, you only have to think differently about the computer you already have.

That strikes many people as a bit strange. There is an old saying that "If it walks like a duck and talks like a duck, it is a duck." A trial-and-error machine looks like a computer and behaves like a computer. So why is it a hypercomputer? The simple answer is "Because it can do more than compute." But few people find that answer convincing, so let me suggest some thoughts that may make this idea a bit easier to swallow.

Notice that there is a distinction between a *machine* and the *machinery* of which it is composed. When you define a machine, you specify how the machinery of which it is composed is to be used. So, for example, both a gas stove and an electric stove are stoves (machines) even though they are made up of different components (machinery). And when your table is broken and you eat on the stove, the stove (machinery) becomes a table (machine). Different machinery, same machine on the one hand. Same machinery, different machine on the other.

Turning a computer (machinery) into a trial-and-error machine changes the way the machinery is used, and that changes the machine. As a result it can change the way computer scientists use it.

You can see the effect that the way you look at your machinery can have on what you do, not only on computer science, but on cognitive science too, by looking at the other side of the computational divide. Below the computing machines there is a hierarchy of machines that use the same machinery but are less powerful than the computing machines. (Chomsky, 1956). At the bottom of this hierarchy is the finite automaton and at the top is the computing machine. Both have precisely the same machinery and both behave in precisely the same way. The only difference between them is that the finite automaton's memory is strictly limited in size and the Turing machine's is not.

That doesn't seem like much of a difference, and it seems even less important if you notice that a Turing machine (with its unlimited memory) cannot be built. (There is a strict finite limit on the amount of space available to even the largest computer.) And yet the distinction between these two types (and the many types of abstract machines

between them) has proved useful in both computer and cognitive science. Looking at the computer *as though* it had an unlimited memory (or space) changes what you do with it. And when you use a computer as though it had unlimited time to complete its operations, you ask different kinds of questions about it than when you assume that it has only computably long to finish. You do different kinds of things with them, and you probably develop different kinds of tools to deal with them.

For mathematicians, there are many problems associated with computing in the limit that seem worth investigating. What exactly does it mean for a process to get closer and closer to a correct algorithm? What is the difference between changing a parameter and changing a program? And can we develop a mathematics of computing in the limit that has the power of the calculus?

For psychologists, the "limiting" view of genuine intelligence raises different issues. It is not easy to see how such intelligence can be measured or studied. Because different styles of genuine intelligence can work in quite different domains and come up with quite different theories of the same evidence, it seems unlikely that these abilities can be measured along a single one-dimensional scale. Chances are that intelligence will be better measured along many dimensions and this will make comparing intelligences more difficult. We may switch from trying measure intelligence in order to compare the intelligence of one person to that of another, to trying to measure the intelligence of a single person to see where that person's intelligence is weak and where it is strong. Such measures might help people take advantage of their strengths and work around (and perhaps ameliorate) their weaknesses.

I have already touched upon some of the questions this view might raise for brain scientists. It would be interesting to know how accurate the division of brain intelligence into the levels of feeling, belief, habit and knowledge intelligence is, and it would be interesting to know whether or not the products of these different kinds of intelligence are stored differently in the brain. Brain scientists studying such storage (or memory) have identified dozens of different types. It would be interesting to know to what degree these different types of memory are related to different kinds of intelligence.

And then there are those more practical enterprises, such as education, management and psychotherapy, that work with human intelligence to produce observable – and presumably measurable – results. Whether the distinction between genuine and fake intelligence could help make these enterprises more effective is, perhaps, the ultimate test of its value.

Consider, for example, education. One of the big disputes among educators is the dispute between those who advocate what is called *constructivism* (Students should construct their own knowledge.) and those who advocate what is called *back to basics* (Students should acquire the knowledge of those who have it.). This seems to be a dispute between those who value genuine intelligence and those who value the fake variety.

But if we understood these two types of intelligence better, and better understood how they work together, we might see the merit of developing both kinds of intelligence in children just as there seems to be some merit in developing both kinds in machines. As I said earlier, there is nothing wrong with fake intelligence. We all have a lot of it, and there is a certain sense in which civilization is built on it.

What's wrong with that? Civilized people eat bread baked for them by others and wear clothes made for them by others. Why shouldn't they also use ideas developed for them by others? Newton was boasting about the fakeness of much of his knowledge when he said that he stood on the shoulders of giants.

But we can probably overdo it in education. Much of what people learn in school is like the kind of scripted knowledge that Searle, in the Chinese Room, has of the Chinese language. (One piece of evidence for that is that many of the people who learn Chinese in school end up unable to speak it.) It lets people fake knowledge. Students are very fond of following the scripts their teacher provide for them. It is quicker and easier than developing your own. The problem is that the results can be both shallow and brittle.

A better understanding of genuine and fake intelligence, and how they work together to produce good thinking, might improve education. Fake and genuine intelligence are two different kinds of trees that grow, if I am right, two different kinds of fruit, both of which seem to play important roles in producing the fruit salad of human thinking.

This suggests the possibility that the best education might come from a marriage of constructivism and back-to-basics in which students learn skills – using their somewhat fake intelligence – that they can then put away as beliefs or habits not to be questioned. These beliefs and habits can then be used, much as the rules of arithmetic were used in extrapolating sequences in the example I used earlier, to go into the construction of higher-level knowledge, using genuine intelligence.

Education serves, at least in part, to develop adults who can do the work of the world. If you want to develop the kinds of thinking that workers needed on the assembly lines that Charlie Chaplin parodied in his movie *Modern Times* – in which the workers work like machines – you don't want them thinking independently. You want them to do what they are told to do – to derive their intelligence from their boss.

That is the kind of worker intelligence assumed by Frederick Winslow Taylor's (1947) management style, known as *scientific management*. The boss – who was assumed to have the genuine intelligence – determined the "one best way" to do things, and the workers simply did what their boss told them to do in the way (the "best way") that he told them to do it.

In more modern styles of management, such as W.E. Deming's (1986) *total quality management*, workers are allowed more initiative and encouraged to use their own genuine intelligence. This allows them to adapt what they are doing as the situation changes.

Today's world seems to require both kinds of workers, and one of the jobs of today's education is to develop them. And it might not be a bad idea if education developed both styles of thinking in the same person.

Another application of a theory of intelligence – a word my dictionary defines as "the ability to acquire and apply knowledge" – is in psychiatry where the aim seems often to be to unlearn something. One can try to erase old learning – the results perhaps of old feeling intelligence – or one can try to help the patient acquire new understanding that will allow him or her to replace or supplant the old.

You may recognize, in the layered approach to intelligence, an echo of the ideas of Freud. But hopefully, there is a difference. Because what I have said here can be said with mathematical precision, because we can develop it into theories of human intelligence that make predictions about human behavior, because it suggests ideas that can be used in our attempts to make both computers and people smarter, it can fail. That's the good news.

It's what gives the right to claim that it is scientific.

## **7. Conclusion**

I have suggested that the answer to the question "Can computers be genuinely intelligent?" is "Yes and no." Computers can be genuinely intelligent, but only if we allow them to do more than compute. Why is that surprising?

Where did the idea that all of thinking could be reduced to computing come from? Why would anybody think that we could come to final conclusions about the future – as final as our conclusion that  $2+3=5$  – based on what we've seen in the past? Why would people think that they, or their computers, would never have to change their minds, which is what restricting yourself to computations ties you to?

I suspect that it is because people worry (and not without reason) about keeping an open mind. One reason to worry is that there are times when you simply can't. When you have to compute. If you're driving a car and you see a truck pulling out of a side street in front of you, you'd better come up with an answer to the question "What shall I do?" quickly and with the sense of finality that a computation gives you. If you

decide you can't make up your mind in a situation like that, you are rightly thought to be a bit stupid (and perhaps dead).

Another reason not to keep an open mind is that there are some things you that you could think about if you wanted to, but you don't want to. For example, you can't do a very good job of driving a car if you keep wondering whether the car will continue to turn right if you turn your steering wheel clockwise. If you keep testing your steering wheel to make sure it hasn't suddenly reversed itself, you will not only weave back and forth across the road, but you might get into trouble because you aren't paying enough attention to the oncoming traffic. If you try to keep an open mind about everything, you will so overload your attention-paying capability that you won't be able to think effectively about anything.

Unfortunately, some people seem to conclude that, since they can't keep an open mind about everything, they shouldn't keep an open mind about anything. That accounts for some of the appeal of the computational model of intelligence which closes its "mind" once the computation is finished. Once you've computed a theory of how your steering wheel works, you can forget about it and you can focus your attention on the road. And that's a good idea.

But that doesn't mean that you can't, or shouldn't, keep an open mind about some things. You might, for example, want to keep an open mind about whether or not you've missed a turn.

It is curious how people like to think that their mind is a single system that works in a single way so that they have to use (say) one kind of intelligence for everything.

Some things can probably be done better by computations and others by uncomputations. There is no need to limit ourselves to one or the other.

There is a huge temptation, at the end of a paper like this one – and I am coming to the end – to assure your readers that you know that what you have said is true. But for me to say that here would be hypocritical. I believe that what I have said here is both true and useful. If I didn't, I wouldn't have said it. But I don't know it. So let me suggest that we might want to develop these ideas with an open mind. If they work, let's stick with them. If they don't, let's try adapting them a bit. If they continue to fail, let's change our minds and try something else.

In other words, let's try to act like trial-and-error machines, even if we aren't.

### **Postscript**

One might wonder why Turing (1950) seems to have understood some of these ideas so long ago, when most of us are only now discovering them. Perhaps it is because he developed the ideas on which the distinction between genuine and fake intelligence, between computation and uncomputation are based. And we only learned them from him.

He got them from experience, whereas we got them from his paper – which has served us as a kind of "oracle". That, of course, is one of the problems of standing on the shoulder of giants. The view may be terrific, but the footing can be slippery. Which could be another reason for wanting computers and children to discover at least some of their ideas on their own rather than letting a programmer or a teacher hand them everything on a silver platter.

## References

Bruner, J.S. (1973), *Beyond the Information Given: Studies in the Psychology of Knowing*, New York, NY: W.W. Norton & Company.

Cattell, R.B. (1963), 'Theory of fluid and crystallized intelligence: A critical experiment', *Journal of Educational Psychology*, 54, pp. 1-22.

Chomsky, N. (1956), 'Three models for the description of language', *IRE Transactions on Information Theory*, IT2, pp. 113-124.

Chomsky, N. (1957), *Syntactic Structures*, The Hague: Mouton & Co.

Chomsky, N. (1957a), 'Review of Skinner', *Language*, 35, pp. 26-58.

Copeland, B.J and Proudfoot, D. (1999), 'Alan Turing's forgotten ideas in computer science', *Scientific American*, 280, pp. 99-103.

Damasio, A.R. (1995), *Descartes' Error: Emotion, reason and the human brain*, New York, NY: G.P. Putnam.

Damasio, A.R., H. Damasio, G.W. Van Hoesen (1982), 'Proropagnosia: anatomic basis and behavioral mechanisms', *Neurology*, 32, 331-341.

Dekker, J.C.E. (1955), 'Productive sets', *Transactions of the American Mathematical Society*, 5, pp. 791-796.

Deming, W.E. (1986), *Out of the Crisis*, Cambridge, MA: MIT, Center of Advanced Engineering Studies.

Eichenbaum, H. and J.A. Bodkin (2000), 'Belief and knowledge as distinct forms of memory', pp. 176-207, in (D.L. Schacter and E. Scarry editors), *Memory, Brain, and Belief*, Cambridge, MA: Havard Press.

Ellis, H.D. and Young, A.W. (1990), 'Accounting for delusional misidentifications', *British Journal of Psychiatry*, 157, pp. 239-248.

Gardner, H. (1993), *Frames of Mind: The theory of multiple intelligences*, New York, NY: Basic Books.

Goedel, K. (1931), 'Ueber formal unentscheidbare Saetze der Principia Mathematica und verwandter System I', *Montascheft Math. Phys.* 38, 173-98.

Gold, E.M. (1965) 'Limiting recursion', *Journal of Symbolic Logic*, 30, 28-48.

Goodman, N., (1954) *Fact, Fiction, and Forecast*, Cambridge, MA: Harvard University Press.

Jain, S., D. Osherson, J.S. Royer, A. Sharma, (1999) *Systems That Learn: An Introduction to Learning Theory*, Second edition, Cambridge, MA: MIT Press.

Kugel, P. (1977) 'Induction pure and simple', *Information and Control*, 35, 276-336.

Kugel, P. (1986) 'Thinking may be more than computing', *Cognition*, 22, 137-198.

Kugel, P. (1990) 'Myhill's thesis: there's more than computing in musical thinking', *Computer Music Journal*, 14, (1) 12-25.

Lucas, J.R., (1961) 'Minds, machines and Goedel', *Philosophy*, Vol. 36, 112-127.

McCarthy, J. (1956), 'The inversion of functions defined by Turing machines', in C.S. Shannon (ed), *Automata Studies*, Princeton University Press.

Martin, E. and D. Osherson, (1998) *Elements of Scientific Inquiry*, Cambridge, MA: MIT Press.

Myhill, J. (1952) 'Some philosophical implications of mathematical logic: three classes of ideas', *Review of Metaphysics* 6 (2), 165-198,

Putnam, H. (1965) 'Trial and error predicates and the solution of a problem of Mostowski', *Journal of Symbolic Logic*, 20, 49-57.

Rosser, J.B. (1936), 'Some extensions of some theorems of Goedel and Church', *Journal of Symbolic Logic*, 1, pp. 87-91.

Scoville, W.B. and B. Milner (1957), 'Loss of recent memory after bilateral hippocampal lesions', *Journal of Neurology, Neurosurgery and Psychiatry*, 20, pp.11-19.

Searle, J. (1980), 'Minds, brains, and programs', *Behavioral and Brain Sciences*, 3, pp. 417-424.

Selfridge, O. (1959), 'Pandemonium: A paradigm for learning', in D. Blake and A. Uttley (eds.), *Proceedings on the Mechanisms of Thought Processes*, HMSO, London.

Taylor, F.W. (1947), *The Principles of Scientific Management*, New York: Norton.

Turing, A.M. (1936) 'On computable numbers, with an application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society, Series 2*, 42, 232-265.

Turing, A.M. (1939) 'Systems of logic based on ordinals', *Proceedings of the London Mathematical Society*, (2) 45, 161-228.

Turing, A.M. (1950) 'Computing machinery and intelligence', *Mind* 59 (N.S. 236), 433-460.

Turing, A.M. (1969), 'Machine intelligence', in B. Meltzer and D. Mitchie (Editors) *Machine Intelligence* 5, 3-23.

Turing, A.M. (1986), 'Lecture to the London Mathematical Society on 20 February 1947', in B.E. Carpenter and R.N. Doran (Editors), *A.M. Turing's ACE Report and Other Related Papers*, Cambridge, MA: MIT Press.

Turing, A.M. (1986a), 'Proposal for Development in the Mathematics Division of an Automatic Computing Engine (ACE)', *ibid.*

Wittgenstein, L. (1958) *Philosophical Investigations*, New York: McMillan.