# Creativity, the Turing Test, and the (Better) Lovelace Test[*]

Selmer Bringsjord & Paul Bello
The Minds & Machines Laboratory
Dept. of Philosophy, Psychology & Cognitive Science
Department of Computer Science
Rensselaer Polytechnic Institute (RPI)
Troy NY 12180 USA
selmer@rpi.edu • bellop@rpi.edu

David Ferrucci
T. J. Watson Research Center
Yorktown Heights, NY 10598
ferrucci@us.ibm.com

5.8.00

**Abstract**

The Turing Test (TT) is claimed by many to be a way to test for the presence, in computers, of such "deep" phenomena as thought and consciousness. Unfortunately, attempts to build computational systems able to pass TT (or at least restricted versions of this test) have devolved into shallow symbol manipulation designed to, by hook or by crook, trick. The human creators of such systems know all too well that they have merely tried to *fool* those people who interact with their systems into believing that these systems really have minds. And the problem is fundamental: the structure of the TT is such as to cultivate tricksters. A better test is one that insists on a certain restrictive epistemic relation between an artificial agent (or system) $A$, its output $o$, and the human architect $H$ of $A$ — a relation which, roughly speaking, obtains when $H$ cannot account for how $A$ produced $o$. We call this test the "Lovelace Test" in honor of Lady Lovelace, who believed that only when computers *originate* things should they be believed to have minds.

## 1 Introduction

As you probably know, Turing predicted in his "Computing Machinery and Intelligence" (1964) that by the turn of the century computers would be so smart that when talking to them from a distance (via email, if you will) we would not be able to tell them from humans: they would be able to pass what is now known as the Turing Test (TT).[1] Well, New Year's Eve of 1999 has come and gone, all the celebratory pyrotechnics have died, and the fact is: AI hasn't managed to produce a computer with the conversational punch of a toddler.

But the really depressing thing is that though progress toward Turing's dream is being made, it's coming only on the strength of clever but shallow trickery. For example, the human creators of artificial agents that compete in present-day versions of TT (like those seen at the Dartmouth conference at which a precursor to this paper was presented) know all too well that they have merely tried to *fool* those people who interact with their agents into believing that these agents really

[*] We are indebted to Jim Moor, Saul Traiger, Jack Copeland, Doug Lenat, and many others who attended the Turing 2000 conference at Dartmouth.

[1] Actually, some understand Turing's prediction to be a more circumspect one, viz., that by 2000 we wouldn't have more than a 70% chance of making a person/machine determination in five minutes.

have minds. The agents in question seem to fall prey, utterly and completely, to Searle's (1980) well-known Chinese Room Argument: These agents are *designed* by their creators to mindlessly manipulate symbols that are perceived by "naive" observers to be indicative of an underlying mind — but "underneath" there is little more than Searle's infamous rulebook. In such scenarios it's really the human creators against the human judges; the intervening computation is in many ways simply along for the ride.

It seems to us that a better test is one that insists on a certain restrictive epistemic relation between a an artificial agent $A$, its output $o$, and the human architect $H$ of $S$ — a relation which, roughly speaking, obtains when $H$ cannot account for how $A$ produced $o$. We call this test the "Lovelace Test" in honor of Lady Lovelace, who believed that only when computers *originate* things should they be believed to have minds.

Our plan herein is as follows. In section 2 we explore Lovelace's complaint in more detail, and we discuss both Turing's first response (from Turing 1964) and a like-minded one given recently by the roboticist Hans Moravec. As you'll see, both responses are anemic. Section 2 also refutes Turing's second response to Lovelace, in which he points out that computers surprise him. Section 3 is devoted to getting on the table a workable characterization of the Lovelace Test. In section 4 we subject three artificial agents to the Lovelace Test. The trio we select fall into the category of those agents intended by their designers to be, in some sense, creative. (Clearly, creative computer systems would have the best chance of avoiding the complaint that a system competing in TT is simply following shallow symbol manipulation tricks devised by its human creator.) One of the three agents was designed and built by Bringsjord and Ferrucci; it is known as BRUTUS. The other two systems, LETTER SPIRIT and COPYCAT, are "creative" systems designed by Douglas Hofstadter, probably the world's leading authority on computational creativity. In section 5 we refute the *third* response Turing gives (again in Turing 1964) to Lovelace — a response that appeals to neural net-based "child machines." In section 6, we consider a final possibility for a system that can move beyond shallow symbol manipulation toward passing LT: oracle machines. We show that such machines still fall prey to Searle's CRA: they do nothing more than mindlessly manipulate symbols in accordance with instructions, and hence fail to do what the LT demands: viz., think for themselves. In the final section, 7, we briefly describe what may be the moral of the story: in order to pass LT and think for itself, a system may have "free will" in the "agent causation" sense. If this is right, it will be rather difficult to build an LT-passing machine.

## 2 Lovelace's Objection from Origination

Lady Lovelace's was perhaps the most powerful objection pressed against TT. Paraphrased, it runs like this:

> Computers can't create anything. For creation requires, minimally, *originating* something. But computers originate nothing; they merely do that which we order them, via programs, to do.[2]

How does Turing respond? Well, at best, mysteriously; at worst, incompetently. Lady Lovelace refers here (in her memoirs) to Babbage's Analytical Engine, which Turing gladly admits did not have the capacity to, as he puts it, "think for itself." So Turing concedes that insofar as Lovelace's argument refers to this device, it goes through. But the property of thinking for itself or of originating something is a property Turing assumes to be possessed by *some* discrete state machines, that is, by some computers — ones that arrived *after* Lovelace passed away. Suppose that

[2] Scholars take note: We have paraphrased Lady Lovelace in a way that implies her position to be that computers *only* do that which we order them, via programs, to do. See Turing's footnote 4 on p. 29 of (Turing 1964).

$M$ is such a machine. Turing then points out that the Analytical Engine was actually a *universal* digital computer, so if suitably programmed, it could perfectly simulate $M$. But such a simulation would bestow upon the Analytical Engine the ability to originate.

Turing's reasoning here is amazingly bad, for the simple reason that Lovelace would hardly have accepted the assumption that such an $M$ exists. What machine did Turing have in mind? What machine fits the bill? He doesn't tell us, but the fact is that the best he and his contemporaries had to offer were machines whose crowning achievements were merely arithmetical.

Next, Turing inexplicably recasts Lovelace's argument as one for the proposition that computers don't superficially surprise us (pp. 21–22 of Turing 1964) — and he then relates what he takes to be an immediate refutation, viz., "Machines take me by surprise with great frequency." Turing's response here has been recently recast by Hans Moravec, who believes that by 2040 not only will TT be passed, but robots will pass the *Total* TT (TTT) as well. (In TTT, due to Stevan Harnad (1991), a robot passes if it is linguistically *and* physically indistinguishable from a human person.) Here is what Moravec says:

> Lady Lovelace, the first programmer, never had a working computer to trouble her programs. Modern programmers know better. Almost every new program misbehaves badly until it is laboriously debugged, and it is never fully tamed. Information ecologies like time-sharing systems and networks are even more prone to wild behavior, sparked by unanticipated interactions, inputs, and attacks. ((Moravec 1999), p. 85)

This is a terribly weak rejoinder. Sure, we all know that computers do things we don't intend for them to do. But that's because we're not smart and careful enough, or — if we're talking about rare hardware errors — because sometimes microscopic events unfold in unforeseen ways. The unpredictability in question does *not* result from the fact that the computer system has taken it upon itself to *originate* something. To see the point, consider the assembling of your Toyota Camry. Suppose that while assembling a bumper, a robot accidentally attaches a spare tire to the bumper instead of leaving it to be placed in its designated spot in the trunk. The cause of the error, assume, is either a fluke low-level hardware error or a bug inadvertently introduced by some programmers. And suppose for the sake of argument that as serendipity would have it, the new position for the tire strikes some designers as the first glorious step toward an automobile that is half conventional sedan and half sport utility vehicle. Would we want to credit the malfunctioning robot with having *originated* a new auto? Of course not.

Things are no different if we consider the specific relationship that impresses Turing and Moravec, namely, the relationship between programmers and their misbehaving programs. Since the three of us both regularly program and regularly *teach* programming, we may not be positioned badly to evaluate this relationship.

It seems to us that programs that surprise because of mere syntactic errors would not be what Turing and Moravec have in mind. To see this, suppose that as part of some larger program $P$ we seek to write a simple Lisp function to triple a given natural number by producing the following code.

```
(defun triple (n)
   (* m 3))
```

Now suppose that at the Lisp prompt `>` we type `(triple 6)` and get back 75. (Of course, a function as trivial as `triple` would in reality be called by another function, but to ease exposition we can assume that we call it directly.) Obviously, ceteris paribus, this will surprise us. What's going on? Well, whereas the argument to the function `triple` is said to be `n` in the argument list in the definition of this function, in the body of the function it's `m`, not `n`, that is multiplied by 3. This slight difference, suppose, was the result of a misplaced keystroke. In addition, though we don't remember doing it, for some (smart, let's assume) reason `m` is elsewhere said to be a global variable whose value is 25.[3]

That this kind of surprise isn't the kind of thing Turing and Moravec have in mind should be beyond doubt. Presumably what they have in mind is a *semantic* bug. What does such a thing look like? We provide an example that will set up later discussion of the BRUTUS system. (This example is used to make a set of different points in (Bringsjord & Ferrucci 2000).)

Suppose that Bill is trying to build a system able to reason about the concept of one person betraying another. And suppose, specifically, that in a rather naive use of first-order logic, Bill has given to this system code that captures this (not entirely implausible) definition:

Def$_B$ 1 Agent $s_r$ betrays agent $s_d$ iff there exists some state of affairs $p$ such that

1 $s_d$ wants $p$ to occur;

2 $s_r$ believes that $s_d$ wants $p$ to occur;

3 $s_r$ agrees with $s_d$ that $p$ ought to occur;

4 $s_r$ intends that $p$ *not* occur;

5 $s_r$ believes that $s_d$ believes that $s_r$ intends that $p$ occur.

In fact, here is what the code might look like (along with relevant conditions instantiated for Dave and Selmer, and with the assumption for contradiction that Selmer doesn't betray Dave):

```
set(auto).
formula_list(usable).

% DefB-2 in {\sc otter}:
all x y (Betrays(x,y) <->
         (exists z (Wants(y,z) &
                    Believes(x,Wants(y,z)) &
                    Agrees(x,y,z) &
                    IntendsNot(x,z) &
                    Believes(x,Believes(y,Intends(x,z)))))).

% Pretend facts of the case:
Wants(adave,agraduate).
Believes(aselmer,Wants(adave,agraduate)).
Agrees(aselmer,adave,agraduate).
IntendsNot(aselmer,agraduate).
Believes(aselmer,Believes(adave,Intends(aselmer,agraduate))).

% Assumption for indirect proof:
-Betrays(aselmer,adave).

end_of_list.
```

This is actual code (for the theorem prover known as OTTER).[4] Notice that this code is using first-order logic in a way that just plain shouldn't work. To see why, consider the formula

$$\forall x(P(x) \rightarrow Q(P(x)).$$

[3] Of course, no competent Lisp programmer would use '`m`' in this way. Some kind of mnemonic would doubtless be employed.

[4] OTTER can be obtained at

`http://www-unix.mcs.anl.gov/AR/otter/`

This formula is non-sensical on the standard grammar of first-order logic. The reason is that the antecedent, $P(x)$, in this universally quantified formula must be one that admits of truth or falsity. For the idea is that if it's true (for some instantiation to $x$), then the consequent, namely, that which is to the right of →, must be true. (Put technically, $P(x)$ is an **atomic formula**, not a **term**.) But this implies that the consequent consists of an atomic formula whose argument is itself an atomic formula, and this, again, is ungrammatical and non-sensical in first-order logic.

But let's suppose that Bill has his system run the betrayal code anyway, just for the heck of it. Bill expects to receive some kind of error message. But what will in fact happen? Well, a contradiction will be found, and Bill will be very surprised to find that apparently his system has proved that Selmer betrays Dave. But there is a serious semantic bug here. The bug is that OTTER has reinterpreted parts of the code in such a way that (e.g.) `Believes` is at once a first-order predicate *and* a functor.

So that you are sure to see what's going on here, watch what happens when we give an input file to OTTER containing the formula just isolated, along with the fact that $P(a)$, where $a$ is a constant, and the assumption for indirect proof, $\neg Q(P(a))$. Here is the input file:

```
set(auto).
formula_list(usable).
all x (P(x) -> Q(P(x))).
P(a).
% Assumption for contradiction:
-Q(P(a)).
end_of_list.
```

And here is the proof from the output file:

```
---------------- PROOF ----------------
1 [] -P(x)|Q(P(x)).
2 [] -Q(P(a)).
3 [] P(a).
4 [hyper,3,1] Q(P(a)).
5 [binary,4.1,2.1] $F.
------------ end of proof -------------
```

This is the same semantic bug. OTTER hasn't really proved what the naive human programmer in this case is seeking. `P` is a functor in line 4 of the proof, but a predicate in line 3. The same phenomenon takes place in the case of Bill.

Now that we have an example of a semantic bug giving rise to surprise, let's ask the key question: Does the fact that Bill's system has surprised him in this way constitute reason for him (or us) to hold that this system can originate anything? Not at all, clearly.

How could Turing and Moravec have missed the mark so badly? Pondered as charitably as possible, this question leads us to surmise that they had in mind a sense of surprise that is deeper than the words they used. (On this reading of them, their examples would simply be regarded as panifully poor, for these examples, by any metric of such things, show exceedingly mundane surprises.) That is, we assume that when surprised in the shallow ways they describe, they *felt* as if they were in fact *deeply* surprised. If when walking around a blind corner, you suddenly spring out and shout, you may surpise one of us to the very core, emotionally speaking — despite the fact that your behavior isn't exactly subtle. As a way to capture a species of surprise that approaches the feelings of Turing's test-minded heart, we suggest a variation on the TT. We call this variation the Lovelace Test (LT).

## 3 The Lovelace Test

To begin to see how LT works, we start with a scenario that is close to home for Bringsjord and Ferrucci, given their sustained efforts to build story generation agents: Assume that Jones, a human AInik, attempts to build an artificial computational agent $A$ that doesn't engage in conversation, but rather creates stories — creates in the Lovelacean sense that this system *originates* stories. Assume that Jones activates $A$ and that a stunningly belletristic story $o$ is produced. We claim that if Jones cannot explain how $o$ was gemerated by $A$, *and* if Jones has no reason whatever to believe that $A$ succeeded on the strength of a fluke hardware error, etc. (which entails that $A$ can produce other equally impressive stories), then $A$ should at least provisionally be regarded genuinely creative. An artificial computational agent passes LT if and only if it stands to its creator as $A$ stands to Jones.

LT relies on the special epistemic relationship that exists between Jones and $A$. But 'Jones,' like '$A$,' is of course just an uninformative variable standing in for any human system designer. This yields the following rough-and-ready definition.

Def$_{LT}$ 1 Artificial agent $A$, designed by $H$, passes LT if and only if

1 $A$ outputs $o$;

2 $A$'s outputting $o$ is not the result of a fluke hardware error, but rather the result of processes $A$ can repeat;

3 $H$ (or someone who knows what $H$ knows, and has $H$'s resources[5]) cannot explain how $A$ produced $o$.

Notice that LT is actually what might be called a *meta*-test. The idea is that this scheme can be deployed for any partcular domain. If conversation is the kind of behavior wanted, then merely stipulate that $o$ is an English sentence (or sequence of such sentences) in the context of a converation (as in, of course, TT). If the production of a mathematical proof with respect to a given conjecture is what's desired, then we merely set $o$ to a proof.

Obvious questions arise at this point. Three that many have asked us upon hearing ancestors of this paper are:

Q1 What resources and knowledge does $H$ have at his or her disposal?

Q2 What sort of thing would count as a successful explanation?

Q3 How long does $H$ have to cook up the explanation?

The answer to the third question is easy: $H$ can have as long as he or she likes, within reason. The proffered explanation doesn't have to come immediately: $H$ can take a month, months, even a year or two. Anything longer than a couple of years strikes us as perhaps unreasonable. We realize that these temporal parameters aren't exactly precise, but then again we should not be held to standards higher than those pressed against Turing and those who promote his test and variants thereof.[6] The general point, obviously, is that $H$ should have more than ample time to sort things out.

But what about Q1 and Q2? Well, these are rather difficult queries. To answer them, we need to explicate the term 'artificial agent,' which stands at the very heart of AI. Fortunately, as the century

[5] For example, the substitute for $H$ might be a scientist who watched and assimilated what the designers and builders of $A$ did every step along the way.

[6] In (Bringsjord 1995), Bringsjord refutes propositions associated with TT by assuming for the sake of argument that some reasonable parameters $\pi$ have been established for this test. But Turing didn't specify $\pi$, and neither have his present-day defenders.

turns, all of AI has been to an astonishing degree unified around a particular conception — that of an intelligent agent. The unification has in large part come courtesy of a comprehensive textbook intended to cover literally *all* of AI: Russell and Norvig's (1994) *Artificial Intelligence: A Modern Approach* (*AIMA*), the cover of which also displays the phrase "The Intelligent Agent Book." The overall, informal architecture for an intelligent agent is shown in Figure 1; this is taken directly from the *AIMA* text. According to this architecture, agents take percepts from the environment, process them in some way that prescribes actions, perform these actions, take in new percepts, and continue in the cycle.[7] In LT, the artificial agent's actions, of course, consist in producing outputs covered by the variable $o$.

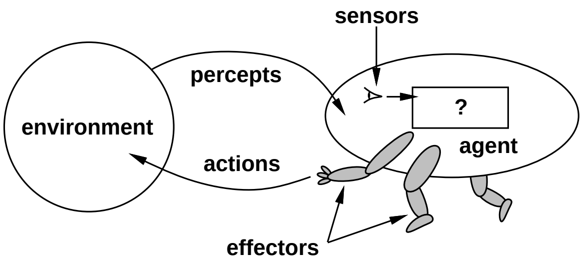


Figure 1: The Architecture of an Intelligent Agent

In *AIMA*, intelligent agents fall on a spectrum from least intelligent to more intelligent to most intelligent. The least intelligent artificial agent is a "TABLE-DRIVEN-AGENT," the program (in pseudo-code) for which is shown in Figure 2. Suppose that we have a set of actions each one of which is the utterance of a color name ("Green," "Red," etc.); and suppose that percepts are digital expressions of the color of an object taken in by the sensor of a table-driven agent. Then given Table 1 our simple intelligent agent, running the program in Figure 2, will utter (through a voice synthesizer, assume) "Blue" if its sensor detects 100. Of course, this is a pretty dim agent. Any output from such an agent will be easy for the designer of a such an agent to explain. So what about smarter agents, ones that might in general be candidates for passing LT?

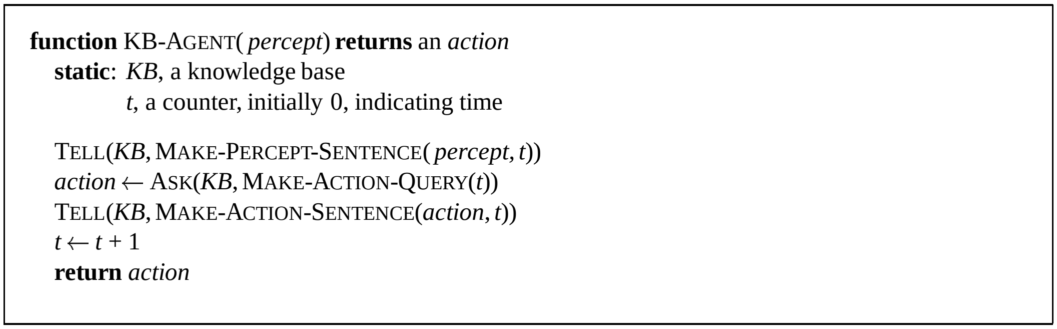
```
function TABLE-DRIVEN-AGENT(percept) returns action
  static: percepts, a sequence, initially empty
          table, a table, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(percepts, table)
  return action
```

Figure 2: The Least Intelligent Artificial Agent

In *AIMA* we reach artificial agents that might strike some as architecturally rich enough to tackle LT when we reach the level of a "knowledge-based" agent. The program for such an agent is shown in Figure 3. This program presupposes an agent that has a knowledge-base (*KB*) in which what the agent knows is stored in formulae in first-order logic, and the functions

- TELL, which injects formulae (representing facts) into *KB*;
- MAKE-PERCEPT-SENTENCE, which generates a first-order formula from a percept and the time $t$ at which it is experienced; and

[7] The cycle here is strikingly similar to the overall architecture of cognition described by Pollock (1995).



```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  $t \leftarrow t + 1$
  return action
```

Figure 3: Program for a Generic Knowledge-Based Agent

Table 1: Lookup Table for TABLE-DRIVEN-AGENT

| **Percept** | **Action** |
|---|---|
| 001 | "Red" |
| 010 | "Green" |
| 100 | "Blue" |
| 011 | "Yellow" |
| 111 | "Black" |

- MAKE-ACTION-SENTENCE, which generates a declarative fact (in, again, the predicate calculus) expressing that an action has been taken at some time $t$

which give the agent the capacity to manipulate information in accordance with first-order logic.

This little tutorial on the nature of intelligent artificial agents allows us to at least make appreciable progress toward answering Q1 and Q2, as follows.

Let's start with

Q1 What resources and knowledge does $H$ have at his or her disposal?

The answer is that $H$ is assumed to have at her disposal knowledge of the architecture of the agent in question, knowledge of the KB of the agent, knowledge of how the main functions in the agent are implemented (e.g., how TELL and ASK are implemented), and so on. $H$ is also assumed to have resources sufficient to pin down these elements, to "freeze" them and inspect them, and so on. We confess that this isn't exactly precise. To clarify things, we offer an example. This example is also designed to provide an answer to the second question, which as you'll recall was

Q2 What sort of thing would count as a successful explanation?

To fix the context for the example, suppose that the output from our artificial agent $A'$ is a resolution-based proof which settles a problem which human mathematicians and logicians have grappled unsuccessfully with for decades. This problem, suppose, is to determine whether or not some formula $\phi$ can be derived from some (consistent) axiom set $\Gamma$. Imagine that after many years of fruitless deliberation, a human $H'$ encodes $\Gamma$ and $\neg\phi$ and gives both to OTTER, and OTTER produces a proof showing that this encoding is inconsistent, which establishes $\Gamma \vdash \phi$, and leads to an explosion of commentary in the media about "brilliant" and "creative" machines, and so

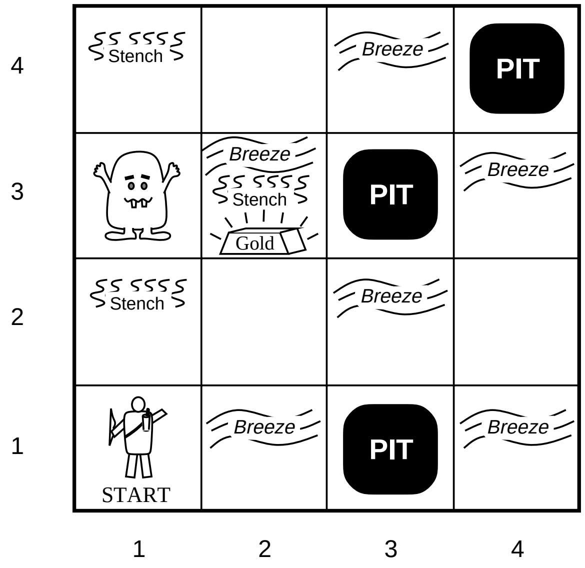


Figure 4: A Typical Wumpus World

on.[8] In this case, $A'$ doesn't pass LT. This is true because $H$, knowing the KB, architecture, and central functions of $A'$ will be able to give a perfect explanation for the behavior in question. This explanation will in principle be no different than explaining the OTTER proof seen above; it will just take longer. One of us (Bringsjord) routinely gives explanations of this sort. The KB is simply the encoding of $\Gamma \cup \{\phi\}$, the architecture consists in the search algorithms used by OTTER, and the main functions consist in the rules of inference used in a resolution-based theorem prover. Put in terms of the Chinese Room, $A$ here doesn't pass LT because $H$, knowing what she knows, could manipulate symbols in accordance with this knowledge and produce the proof in question.

Here, now, given the foregoing, is a better definition:

Def$_{LT}$ 2 Artificial agent $A$, designed by $H$, passes LT if and only if

1 $A$ outputs $o$;

2 $A$'s outputting $o$ is not the result of a fluke hardware error, but rather the result of processes $A$ can repeat;

3 $H$ (or someone who knows what $H$ knows, and has $H$'s resources) cannot explain how $A$ produced $o$ by appeal to $A$'s architecture, knowledge-base, and core functions.

## 4 How do Today's Systems Fare in the Lovelace Test?

Today's systems, even those designed to either be, or seem to be, creative, fail LT. They are all systems whose designers can easily imagine "Chinese Roomifying," that is, these designers can imagine themselves generating the output in question by merely manipulating symbols in accordance with the knowledge bases, algorithms, and code in question. We give three examples of this kind of failure.

[8] For a "real life" counterpart, we have OTTER's settling the Robbins Problem, presented as an open question in (Wos 1996).

### 4.1 BRUTUS

Let's turn first to the BRUTUS system (Bringsjord & Ferrucci 2000). This is a system designed to appear to be literarily creative to *others*. To put the point in the spirit of TT, BRUTUS reflects a multi-year attempt to build a system able to play the short short story game, or S$^3$G for short (Bringsjord 1998*a*). (See Figure 5 for a picture of S$^3$G.)

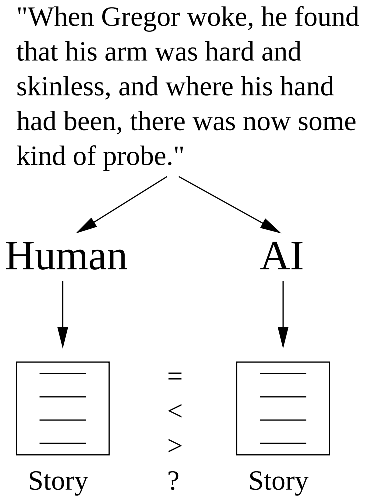

"When Gregor woke, he found that his arm was hard and skinless, and where his hand had been, there was now some kind of probe."
Human
AI
=
<
>
?
Story
Story

Figure 5: The Short Short Story Game, or S$^3$G for Short.

The idea behind S$^3$G is simple. A human and a computer compete against each other. Both receive one relatively simple sentence, say: "As Gregor Samsa awoke one morning from uneasy dreams he found himself transformed in his bed into a gigantic insect." (Kafka 1948, p. 67) Both mind and machine must now fashion a short short story (about 500 words) designed to be truly interesting; the more literary virtue, the better. The goal in building BRUTUS, then, is to build an artificial author able to compete with first-rate human authors in S$^3$G, much as Deep Blue went head to head with Kasparov.

How does BRUTUS fare? Relative to the goal of passing S$^3$G, not very well. On the other hand, BRUTUS can "author" some rather interesting stories, e.g.,

"Betrayal in Self-Deception" (conscious)

> Dave Striver loved the university. He loved its ivy-covered clocktowers, its ancient and sturdy brick, and its sun-splashed verdant greens and eager youth. He also loved the fact that the university is free of the stark unforgiving trials of the business world — only this *isn't* a fact: academia has its own tests, and some are as merciless as any in the marketplace. A prime example is the dissertation defense: to earn the PhD, to become a doctor, one must pass an oral examination on one's dissertation. This was a test Professor Edward Hart enjoyed giving.
>
> Dave wanted desperately to be a doctor. But he needed the signatures of three people on the first page of his dissertation, the priceless inscriptions which, together, would certify that he had passed his defense. One of the signatures had to come from Professor Hart, and Hart had often said — to others and to himself — that he was honored to help Dave secure his well-earned dream.
>
> Well before the defense, Striver gave Hart a penultimate copy of his thesis. Hart read it and told Dave that it was absolutely first-rate, and that he would gladly sign it at the defense. They even shook hands in Hart's book-lined office. Dave noticed that Hart's eyes were bright and trustful, and his bearing paternal.

> At the defense, Dave thought that he eloquently summarized Chapter 3 of his dissertation. There were two questions, one from Professor Rodman and one from Dr. Teer; Dave answered both, apparently to everyone's satisfaction. There were no further objections.
>
> Professor Rodman signed. He slid the tome to Teer; she too signed, and then slid it in front of Hart. Hart didn't move.
>
> "Ed?" Rodman said.
>
> Hart still sat motionless. Dave felt slightly dizzy.
>
> "Edward, are you going to sign?"
>
> Later, Hart sat alone in his office, in his big leather chair, saddened by Dave's failure. He tried to think of ways he could help Dave achieve his dream.

Note that we have placed the term 'author' in scare quotes. Why? The reason is plain and simple, and takes us back to Lady Lovelace's objection: BRUTUS didn't *originate* this story. He is capable of generating it because two humans, Bringsjord and Ferrucci, spent years figuring out how to formalize a generative capacity sufficient to produce this and other stories, and they then are able to implement part of this formalization so as to have a computer produce such prose. This method is known as *reverse engineering*. Obviously, with BRUTUS set to $A$ and Bringsjord and Ferrucci set to $H$ in the definition of LT, the result is that BRUTUS fails this test. Put in terms of the Chinese Room, both Bringsjord and Ferrucci could take the place of a computer and mindlessly manipulate what they know (algorithms, knowledge-base, etc.) in order to produce output that impresses human observers.

Let's now give you, briefly, a specific example to make this failure transparent. BRUTUS is programmed to produce stories that, are, at least to some degree, bizarre. The reason for this is that reader response research tells us that readers are engaged by bizarre material. Now, in BRUTUS, to express the bizarre, modifiers are linked with objects in frames named `bizzaro_modifiers`. Consider the following instance describing the `bizzaro` modifier `bleeding`.

```
instance bleeding is a bizzaro_modifier
    objects are {sun, plants, clothes, tombs, eyes}.
```

What Bringsjord and Ferrucci call **literary augmented grammars**, or just a LAGs, may be augmented with constraints to stimulate bizarre images in the mind of the reader. The following LAG for action analogies,

- `BizarreActionAnalogy` → `NP VP like ANP`
- `NP` → `noun_phrase`
- `ANP` → `modifier (isa bizzaro_modifier) noun (isa analog of NP)`

in conjunction with `bizzaro_modifiers` can be used by BRUTUS to generate the following sentence.

> *Hart's eyes were like big bleeding suns.*

Sentences like this in output from BRUTUS are therefore a function of work carried out by (in this case) Ferrucci. Such sentences do not result from BRUTUS thinking on its own.

### 4.2 COPYCAT

Douglas Hofstadter, as many readers know, has thought a lot about creativity, and has built systems in order to explore and validate the result of that thought. So what sort of creativity does Hofstadter focus on? And what are the relevant systems? One representative system is COPYCAT, described at length in (Hofstadter 1995). COPYCAT is supposed to solve problems like the following two by coming up with "creative analogies."

Problem 1 Suppose the letter-string ***abc*** were changed to ***abd***; how would you change the letter-string ***ijk*** in "the same way"?

Problem 2 Suppose the letter-string ***aabc*** were changed to ***aabd***; how would you change the letter-string ***ijkk*** in "the same way"? were changed to ***kji*** in

COPYCAT settles in on ***ijl*** as an answer for Problem 1, and in the process "considers" ***ijd*** and ***ijj***. For Problem 2, the program arrives at ***ijll***, and "considers" ***ijkl***, ***jjkk***, ***hjkk***, ***jkkk***, ***ijkd***, ***ijdd***, ***ijkk***, and ***djkk***. Are these good answers? Are they creative? Hofstadter answers "Yes" to both questions. But he seems not to notice that he answers in this way only because COPYCAT has been designed to mirror the answers he (and many other humans) would be inclined to give, and for this reason COPYCAT fails LT. COPYCAT gives the kind of answers it does because rules like "Replace the rightmost letter with its successor" are employed. But what recommends these rules, rather than others which COPYCAT has no "awareness" of? COPYCAT seems thoroughly ad hoc.

As evidence for the capricious nature of COPYCAT's answers, consider the fact that the answers one of us gave, after seeing these problems for the very first time, were ***ijj*** and ***ijkj***. (Notice that the second of these isn't even "considered" by COPYCAT.) The rule that produced these answers was one based on rhyming. In ***abc***, the second and third letters rhyme. When this string is replaced with ***abd***, this rhyme is preserved. In the string ***ijk***, the second and third letters rhyme. To follow the rule in question, ***k*** must be replaced with a different letter that rhymes with ***j***, the second letter. The only possibility is ***j***; hence ***ijj*** is produced. The same rule, for obvious reasons, yields ***ijkj***.

That COPYCAT fails LT is transparent when you consider how the system would look if couched in theorem proving terms. It would seem that the rules of letter-string replacement to which Hofstadter is drawn can be effortlessly expressed as formulas in first-order logic. For example, let $l$ be a function mapping character triples $(c_1, c_2, c_3)$ into letter-strings, so that $l(a, b, c)$ is ***abc***. Now let $s$ be the successor function for the 26 lower-case letters $\{a, b, c, \ldots, z\}$. Then here is a rule for the replacement function $r$:

$$\forall x \forall y \forall z (r(l(x, y, z)) = l(x, y, s(z))).$$

It is easy to capture letter-replacement with rules like this, and to assign weights to these rules. (It can be done with OTTER, the theorem prover discussed above.) Producing a solution would then consist in the production of proofs after starting strings (e.g., ***ijk*** in Problem 1) are given as additional input to the system. Obviously, a designer of the theorem proving version of COPYCAT would know exactly why the system produces the output it does. Or, to couch the point in Searlean terms, the formulae in the theorem proving version could be used as a correlate to the "rulebook" in the Chinese Room, and one of us, using it as such, could get inside a box and fool people on the outside into believing that the system is "thinking for itself."

### 4.3 LETTER SPIRIT

Things are no different when we (briefly) consider LETTER SPIRIT, another system from Hofstadter 1995. Whereas the domains dealt with by COPYCAT and BRUTUS are textual in nature, LETTER

Spirit deals with a visual one. Letter Spirit is so named because it is intended to capture the essence of a letter (columns in Figure 7) and the "spirit" of a font (rows in Figure 7). More specifically, Letter Spirit works within the space of the Roman alphabet.[9] Originally, Letter Spirit was intended to invent new fonts within the space of Roman alphabets, but early on Hofstadter (and his collaborator in this endeavor: McGraw) decided that that was too difficult. So they decided instead to have Letter Spirit take in a few "seed" letters in a certain font as percepts, and yield as output the remaining letters in this font; and they decided that the fonts in question would all have to come from a restricted class of fonts, "gridfonts" as Hofstadter and McGraw call this class. (Each gridfont letter is created by selecting from among 56 quanta. See Figure 8.) This retreat is a pity, for it would seem that an agent able to cook up brand new fonts in the unrestricted space of the Roman alphabet might have a chance at passing LT. But Letter Spirit has no such chance. The reason is that Hofstadter and McGraw can explain exactly how it is that Letter Spirit gives the remaining letters in a particular font, by appeal to the relevant ingredients (knowledge-base, central algorithms, etc.). In Searlean terms, the two of them, upon being given some seed letters, can use these ingredients to mindlessly complete the font.[10]

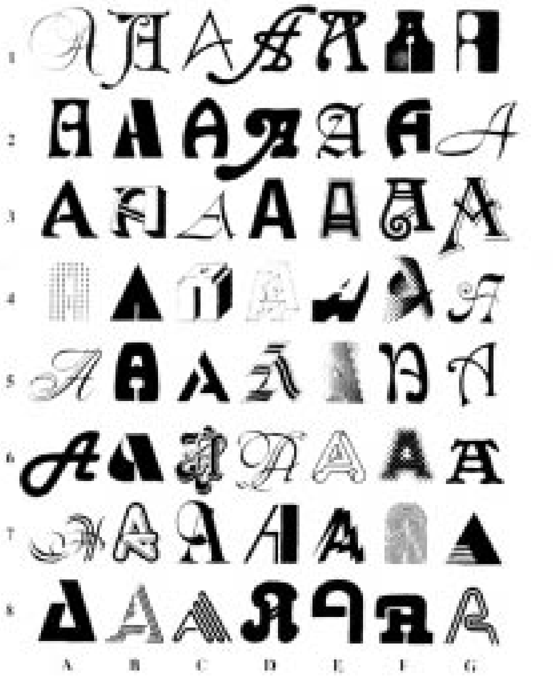


Figure 6: Various Letter As

At this point it's time to consider the objection no doubt many of our readers are itching to press against us.

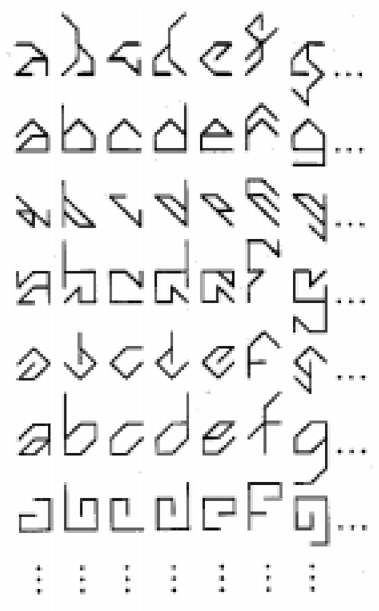


Figure 7: “Letter” is Covered by Columns, “Spirit” by Rows

## 5 The Objection From Learning

Here is how the objection in question runs: “Gentlemen, this much is sure: You will never win any awards as Turing scholars. For you conveniently neglected to note earlier that just before recasting Lady Lovelace’s argument as one aimed at substantiating the view that computers never surprise us, there is a paragraph consisting of a lone sentence, namely: ‘The whole question will be considered again under the heading of learning machines.’ (Turing 1964, p. 21) By ‘whole question’ Turing is referring to none other than the question with which the three of you, following Lovelace, are concerned, that is, whether or not a computer can originate anything. The heading in question is “Learning Machines,” and it stands atop the final section, 7, of Turing’s seminal paper. What Turing says in that section destroys Lovelace’s complaint, and likewise it destroys your own complaint that machines originate nothing. In fact, modern-day embodiments of what Turing envisions in this section, namely connectionist neural network systems, pass your supposedly grueling LT with flying colors. Let me explain.

“Turing’s strategy for building a machine capable of passing TT is *not* to program a machine from scratch, injecting knowledge (and, yes, trickery) into it. His strategy, expressed in the “Learning Machines” section, is instead to first build what he calls a “child-machine,” and to then teach it in much the same way that we teach our own youth. Here is a quote that hits you two rather hard.”

> An important feature of the learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil’s behavior. This should apply most strongly to the later education of a machine arising from a child-machine of well-tried design (or program). This is in clear contrast with normal procedure when using a machine to do computations: one’s object is then to have a clear mental

[9] A part of a part of this space is shown in Figure 6. By Hofstadter’s lights, the space in question may be uncountable: see (Hofstadter 1982). For a discussion of this possibility, see (Bringsjord & Zenzen 2001).

[10] This comes as no surprise given the method used by Hofstadter and McGraw: They based LETTER SPIRIT on studies of how humans create and complete gridfonts. Some human-created gridfonts are shown in Figure 9.

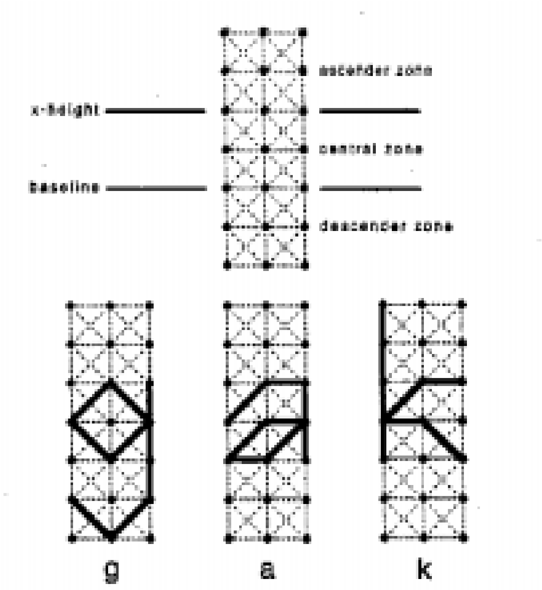


Figure 8: The 56 Quanta Defining Gridfonts

> picture of the state of the machine at each moment in the computation. This object can only be achieved with a struggle. The view that "the machine can only do what we know how to order it to do" appears strange in the face of this. (Turing 1964, p. 29)

This objection is actually rather easily surmounted, as follows. Presumably the child-machine in question develops its abilities by virtue of training of artificial neural networks (ANNs). Now, the information processing of an ANN corresponds in all cases to standard computation (e.g., Turing machine computation), or standard deduction (e.g., proofs in first-order logic), as Bringsjord (1991) has explained elsewhere. To fix this point we can appeal to the notion of **representability**, used in some contemporary proofs of Gödel's first incompleteness theorems (e.g., see section §7 of Chapter X of Ebbinghaus, Flum & Thomas 1984). Here is the relevant definition:[11]

> A function $f : \mathbf{N}^m \rightarrow \mathbf{N}$ is called **representable** in a set $\Phi$ of first-order formulas about arithmetic iff there is a formula $\phi(v_1, \ldots, v_{m+1})$ (i.e., a formula whose free variables are only $v_1, \ldots, v_{m+1}$) about arithmetic such that for all $n_1, \ldots, n_{m+1} \in \mathbf{N}$,
>
> - if $f(n_1, \ldots, n_m) = n_{m+1}$ then $\Phi \vdash \phi(\tilde{n_1}, \ldots, \tilde{n_{m+1}})$
> - if $f(n_1, \ldots, n_m) \neq n_{m+1}$ then $\Phi \vdash \neg\phi(\tilde{n_1}, \ldots, \tilde{n_{m+1}})$
> - $\Phi \vdash \exists^{=1} v_{m+1}\, \phi(\tilde{n_1}, \ldots, \tilde{n_m}, v_{m+1})$
>
> Here we say that $\phi(v_1, \ldots, v_{m+1})$ **represents** $f$ in $\Phi$.

Now, suppose that the child-machine $M_c$, on the strength of ANNs, computes some function $f$. This function is representable in some $\Phi$. You can think of $\Phi$ in this case as a knowledge-base. But then there is no longer any "thinking for itself" going on, for if we assume a computer scientist to be in command of the knowledge-base $\Phi$ and the relevant deduction from it, the reasons for this scientist to declare the child-machine a puppet are isomorphic to the reasons that compel the designer of knowledge-based systems like BRUTUS to admit that such a system originates nothing.

[11] We simplify this definition for the present context. The phrase "about arithmetic" is of course made precise in the formal version. A tilde above a character indicates that that character is one *in* the logic, so $\tilde{7}$ is a constant used in the logic to refer to the number 7.

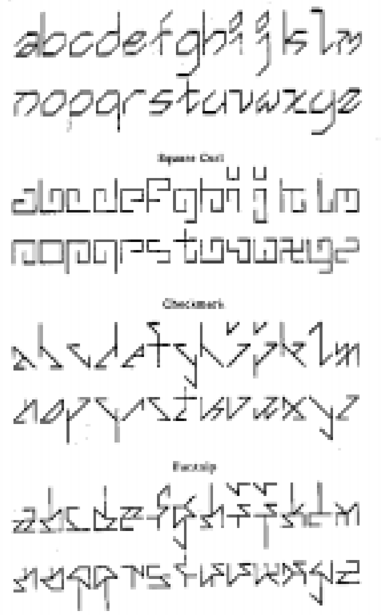


Figure 9: Some Human-Designed Gridfonts

# 6 Do Oracle Machines Pass the Lovelace Test?

So what kind of system *can* pass LT? How can a system break the bounds of mere symbol manipulation to do something on its own? It might be thought that so-called "oracle machines" can pass LT; here is how this idea would run: Jack Copeland (1998) has recently argued that oracle machines (or, as he calls them, 'O-machines') are immune to Searle's (1980) CRA-based claim that mere symbol manipulation is insufficient for genuine mentation. Since those systems that fail LT appear to be failing because it's clear that they are merely manipulating symbols in predictable ways, the idea is that oracle machines might pass LT because they do more than manipulate symbols.

Unfortunately, oracle machines are *not* immune to the Chinese Room. Copeland's characterization of these machines is just plain wrong. When one is clear and correct about oracle machines, it becomes obvious that they fall prey to the Chinese Room. And given that, it becomes obvious as well that they provide nothing that might help to pass LT.

## 6.1 Getting Straight About Oracle Machines

Copeland's account of O-machines should prove to be a shocker to those logicians and mathematicians familiar with the formal terrain in question. He tells us that O-machines are "digital computing machines" (Copeland 1998, p. 128); he says that

> They generate digital output from digital input by means of a step-by-step procedure consisting of repeated applications of a small, fixed number of primitive operations, the procedure unfolding under the control of a finite program of instructions which is stored internally in the form of data on the machine's tape. (Copeland 1998, p. 128–129)

This is just simply false. Again, the 'O' in 'O-machine' stands for '*oracle*'; these machines, accordingly, work by way of oracles. They mark a way of creating, in one avowedly mysterious stroke, a portal to an interesting realm: the realm which opens up once problems proved to be mechanically unsolvable (= unsolvable by Turing machines = beyond the so-called Turing Limit) are assumed to be *somehow* solvable. Oracle-machines are simply Turing machines augmented with an oracular ability to solve the halting problem; they are nothing more. They are not computing machines. They don't work exclusively "by means of a step-by-step procedure consisting of repeated applications of a small, fixed number of primitive operations." No such procedure exists for solving the halting problem; that's why the problem is said to be unsolvable.

Textual confirmation of the claim that Copeland is just fundamentally mistaken about the nature of O-machines is easy to come by. Here is how a recently updated classic textbook on computability and uncomputability introduces oracles:

> Once one gets used to the fact that there are explicit problems, such as the halting problem, that have no algorithmic solution, one is led to consider questions such as the following:
>
> Suppose we were given a "black box" or, as one says, an *oracle*, which can tell us whether a given Turing machine with given input eventually halts. Then it is natural to consider a kind of program that is allowed to ask questions of our oracle and to *use the answers in its further computation* ... (Davis, Sigal & Weyuker 1994: 197; emphasis ours)

Notice what we have emphasized in this quote. The idea is that computation *calls* the oracle; the oracle itself is not part of the computation.

How do Davis et al. transform this figurative scheme into a mathematically respectable one? To answer this question, note that instead of Turing machines, Davis et al. use an equivalent programming language $\mathcal{L}$, the programs of which are composed of lists of statements with optional labels. $\mathcal{L}$ allows for three types of statements: adding one to a variable $V$ ($V \leftarrow V+1$), subtracting one from a variable $V$ ($V \leftarrow V-1$), and moving by a conditional to a line labeled with $L$ in a program (IF $V \neq 0$ GOTO $L$). With just these three statements it is possible to write a program that computes every Turing-computable function. Traditionally, to make it easier to see this, "macros" $V \leftarrow V'$ and GOTO $L$ are allowed. The first macro moves the contents of variable $V'$ to variable $V$; the second is an unconditional branch that moves the active line to the one with label $L$; both macros can be easily decomposed into a program written with only the three fundamental statements. As an example of a simple program in $\mathcal{L}$, consider a program that computes the function $f(x_1, x_2) = x_1 + x_2$:[12]

```
      $Y \leftarrow X_1$
      $Z \leftarrow X_2$
[$B$] IF $Z \neq 0$ GOTO $A$
      GOTO $E$
[$A$] $Z \leftarrow Z - 1$
      $Y \leftarrow Y + 1$
      GOTO $B$
```

Now we're in position to see how Davis et al. formalize oracles. The trick is simply to allow a new statement (an *oracle statement*) of the form

$$V \leftarrow O(V)$$

into the syntax of $\mathcal{L}$. "We now let $G$ be some partial function on $\mathbf{N}$ (the natural numbers) with values in $\mathbf{N}$, and we shall think of $G$ as an oracle" (Davis et al. 1994: 198). So if the value of variable $V$ is $m$ before an oracle statement is encountered, when the statement is then reached, the value of $V$ changes to $G(m)$ (assuming that $G$ is defined for this argument). As should be plain, *there*

[12] Note that a conditional or unconditional branch that directs flow to a label not present in the program causes halting. In the program here, then, the label $E$ can be read as "exit."

*is absolutely no sense in which G is computed.* $G$ is just a placeholder for what at this point is, to say it yet again, oracular. In connection, specifically, with the halting problem, where $M_1, M_2, \ldots$ enumerates all Turing machines, the function

$$h(m, n) = \begin{cases} 1 & \text{if } M_m \text{ halts with input } n \\ 0 & \text{otherwise} \end{cases}$$

can be "solved" by a program in $\mathcal{L}$ in which a gödel encoding of $m$ and $n$ is given as an argument to $G$.

How did Copeland go so wrong? How did he come to describe O-machines as standard computing machines? It would seem that Copeland fell prey to two confusions. To see the first, and to feel its pull, note that if you have an oracle on hand, at your beck and call, you can certainly treat your submission of a question to the oracle as a "fixed, primitive" step; and you can treat the return of an answer from the oracle similarly. After all, this is why formalization of "oracle consultation" can be symbolized in the manner we have just seen. But "fixed, primitive steps" should not be confused with computation or calculation, as is easy enough to see: If God exists, presumably He can be petitioned in one simple step, and can return an answer in one simple step as well. But ascertaining whether or not a Turing machine halts courtesy of the Almighty would seem to exceed computation by just a tad. If God routinely (oracularly) answered the fixed "queries" of Turing machine $M$ with a fixed verdict, we could stipulatively define a composite "O-machine" $M$+God, but this "machine" wouldn't be, to use Copeland's language, a "digital computing machine."

Can the appeal to the oracular be removed in favor of detailed logico-mathematical devices? Yes. In fact, many thinkers occupied with the philosophy and logic of minds and machines have produced and discussed such devices, so it's somewhat odd that Copeland believes O-machines to be "little known among philosophers of mind" (Copeland 1998, p. 129). (As we'll see in moment, the philosopher Hilary Putnam long ago introduced an informative substitution for O-machines.) Indeed, one of us has recently written about a potential explosion in the intersection of philosophy and super-computation (Bringsjord 1998). O-machines, in our experience, are well-known among technical philosophers of mind, but are generally ignored because their role, as we indicated above, is to simply open the portal to processing above the Turing Limit. They provide no substance relative to the powers of minds and machines.

So, what logico-mathematical devices are options, and who has discussed them? Just as there are an infinite number of mathematical devices that are equivalent to Turing machines (machines running programs from the language $\mathcal{L}$ visited above, Register machines, the $\lambda$-calculus, abaci, ...; these are all discussed in the context of an attempt to define computation in Bringsjord 1994), there are an infinite number of devices beyond the Turing Limit. As you might also guess, a small proper subset of these devices dominate the literature. In fact, three kinds of super-computational devices — analog chaotic neural nets, trial-and-error machines, and Zeus machines — are generally featured in the literature. In the interests of reaching a wider audience, we discuss only the latter two devices here.[13]

Trial-and-error machines have their roots in a paper by Hilary Putnam (1994), and one by Mark Gold (1994); both appeared in the same rather famous volume and issue of the *Journal of Symbolic Logic*. So what *are* trial-and-error machines? Well, they have the architecture of Turing machines (read/write heads, tapes, a fixed and finite number of internal states), but produce output "in the

[13] Analog chaotic neural nets are characterized by Siegelmann & Sontag (1994). For cognoscenti, analog chaotic neural nets are allowed to have irrational numbers for coefficients. For the uninitiated, analog chaotic neural nets are perhaps best explained by the "analog shift map," explicated by Siegelmann (1995) Siegelmann, and summarized in (Bringsjord 1998*b*).

limit" rather than giving one particular output and then halting. Here is a trial-and-error machine $\mathcal{M}$ that solves the halting problem. Take some arbitrary Turing machine $M$ with input $u$; let $n^{M,u}$ be the Gödel number of the pair $M$, $u$; place $n^{M,u}$ on $\mathcal{M}$'s tape. Now have $\mathcal{M}$ print 0 immediately (recall the function $h$, defined above), and then have it simulate the operation of $M$ on $u$. If $\mathcal{M}$ halts during the simulation, have it proceed to erase 0 in favor of 1, and then have it stop for good. It's as easy as that.[14]

Zeus machines (or "Weyl Machines" from (Weyl 1949); see also Bertrand Russell's (1936) discussion of the possibility of his embodying such devices) are based on the character Zeus, described by Boolos & Jeffrey (1989). Zeus is a superhuman creature who can enumerate $\mathbf{N}$ *in a finite amount of time*, in one second, in fact. He pulls this off by giving the first entry, 0, in $\frac{1}{2}$ second, the second entry, 1, in $\frac{1}{4}$ second, the third entry in $\frac{1}{8}$ second, the fourth in $\frac{1}{16}$ second, ..., so that, indeed, when a second is done he has completely enumerated the natural numbers. Obviously, it's easy to adapt this scheme so as to produce a Zeus machine that can solve the halting problem: just imagine a machine which, when simulating an arbitrary Turing machine $M$ operating on input $u$, does each step faster and faster ... (There are countably many Turing machines, and those that don't halt are trapped in an unending sequence of the same cardinality as $\mathbf{N}$.) If, during this simulation, the Zeus machine finds that $M$ halts on $u$, then a 1 is returned; otherwise 0 is given.

### 6.2 Copeland's Argument for Why O-Machines Dodge CRA

Here is Copeland's argument for the claim that Searle's CRA is powerless against a view of the mindbrain according to which it's an O-machine rather than a Turing machine:

> An O-machine's program may call for primitive operations that a human clerk working by rote and unaided by machinery is incapable of carrying out (for otherwise, by the Church-Turing thesis, whatever can be calculated by an O-machine can be calculated by a Turing machine — a contradiction). It follows that there is no possibility of Searle's Chinese Room argument [CRA] being successfully deployed against the new functionalism offered by hypothesis (1a). (Copeland 1998, p. 132)

Alas, this argument fails, as is easy to see. An O-machine, in calling for the oracular, certainly does call for something that a clerk (i.e., Turing's "computists" or Post's "workers") will have a hard time providing. (And Searle, as mere symbol manipulator, will have a similarly hard time simulating an O-machine.) But then again, only an oracle, by definition, can meet the challenge. Copeland's "new functionalism" is really mysticism; and CRA was never billed as a refutation of any such doctrine. When O-machines are specified, Copeland's argument is revealed as invalid. For example, suppose that rather than O-machines we talk of Zeus machines. It's easy to imagine that Searle in the Chinese Room manipulates symbols in order to parallel the operation of a Zeus machine — and doing so would no more guarantee the appropriate phenomenal consciousness (e.g., grasping that when a native Chinese speaker outside the room sends in a squiggle-squoggle he is asking if Searle savors perfectly grilled hamburgers) than would Searle's symbol manipulation in the original gedanken-experiment. The only difference is that in (what we might call) the "Zeus Room," Searle works much faster. But this speed-up makes no difference with respect to true understanding. After all, Zeus could be a pigeon. And a pigeon trained to move symbols around, even if blessed with the ability to carry out this movement at Zeus-level speeds, would still have the mental life of a bird, which of course falls far short of truly understanding Chinese.

[14] For full exposition, along with arguments that human persons are trial-and-error machines, see (Kugel 1986), a seminal paper that situates trial-and-error machines nicely within both the formal context of the Arithmetic Hierarchy and the philosophical context of whether minds are computing machines.

The upshot of this for oracle machines and LT is plain. If one designs a "super pigeon" to produce some output, one will be no more inclined to believe that this bird is thinking for itself than one would be inclined to believe that a slower, "standard" pigeon would have the kind of autonomy Lady Lovelace demands.

## 7 What Then Does it *Take* to Pass LT?

Many readers will doubtless ask: So what do you have to say, *constructively* speaking? So far, everything you've given us is negative; how about something positive? What should AIniks do to build an LT-passing system that breaks the bounds of mindless symbol manipulation to think for itself? We end with some brief speculation arising from such questions.

Unfortunately, it seems to us (or at least to *one* of us: Bringsjord) that the moral of the story may be irredeemably negative. There may simply not be a way for a mere information-processing artifact to pass LT, because what Lovelace is looking for may require a kind of autonomy that is beyond the bounds of ordinary causation and mathematics. The notion that creativity requires autonomy is one anticipated, at least in nascent terms, by Hofstadter (e.g., see pp. 411 of Hofstadter 1995), who seems confident that computation will ultimately be up to the task of capturing the kind of autonomy creative humans exploit. But what if the kind of "thinking for oneself" required by LT entails a form of autonomy known as *agent causation*? The doctrine of agent causation, which is set out, defended, and shown to be beyond ordinary computation in "Chapter VIII: Free Will" of (Bringsjord 1992), entails the view that persons bring about certain states of affairs (e.g., mental events like decisions) directly, with no ordinary physical causal chain in the picture. Though some famous philosophers have affirmed this view (e.g., Richard Taylor and Roderick Chisholm), it hasn't been all that popular. This paper isn't intended to promote agent causation; no argument for the view has been articulated herein. The point in this last paragraph is only to raise the *possibility* that the difficulty we have in conceptualizing (let alone building) an LT-passing artificial agent may inhere in the fact that such an agent must have a rather radical kind of autonomy. To put the point another way, the difficulty in question may provide inductive evidence for the view that agent causation is real in our creative lives.[15]

[15] The point expressed this way bears an interesting resemblance to Moor's (1976) stance on TT.

# References

Boolos, G. S. & Jeffrey, R. C. (1989), *Computability and Logic*, Cambridge University Press, Cambridge, UK.

Bringsjord, S. (1991), 'Is the connectionist-logicist clash one of ai's wonderful red herrings?', *Journal of Experimental & Theoretical AI* **3.4**, 319–349.

Bringsjord, S. (1992), *What Robots Can and Can't Be*, Kluwer, Dordrecht, The Netherlands.

Bringsjord, S. (1995), Could, how could we tell if, and why should–androids have inner lives?, *in* K. Ford, C. Glymour & P. Hayes, eds, 'Android Epistemology', MIT Press, Cambridge, MA, pp. 93–122.

Bringsjord, S. (1998*a*), 'Chess is too easy', *Technology Review* **101**(2), 23–28.

Bringsjord, S. (1998*b*), Philosophy and 'super' computation, *in* J. Moor & T. Bynam, eds, 'The Digital Phoenix: How Computers are Changing Philosophy', Blackwell, Oxford, UK, pp. 231–252.

Bringsjord, S. & Ferrucci, D. (2000), *Artificial Intelligence and Literary Creativity: Inside the Mind of Brutus, a Storytelling Machine*, Lawrence Erlbaum, Mahwah, NJ.

Bringsjord, S. & Zenzen, M. (2001), *SuperMinds: A Defense of Uncomputable Cognition*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Copeland, B. J. (1998), 'Turing's O-machines, searle, penrose and the brain', *Analysis* **58**(2), 128–138.

Ebbinghaus, H. D., Flum, J. & Thomas, W. (1984), *Mathematical Logic*, Springer-Verlag, New York, NY.

Gold, M. (1994), 'Limiting recursion', *Journal of Symbolic Logic* **30**(1), 28–47.

Harnad, S. (1991), 'Other bodies, other minds: A machine incarnation of an old philosophical problem', *Minds and Machines* **1.1**, 43–54. This paper is available online at ftp://cogsci.ecs.soton.ac.uk/pub/harnad/Harnad/harnad91.otherminds.

Hofstadter, D. (1982), 'Metafont, metamathematics, and metaphysics', *Visible Language* **14**(4), 309–338.

Hofstadter, D. (1995), *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*, Basic Books, New York, NY.

Hofstadter, D. & McGraw, G. (1995), Letter spirit: Esthetic perception and creative play in the rich microcosm of the roman alphabet, *in* 'Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought', Basic Books, New York, NY, pp. 407–488.

Kafka, F. (1948), The metamorphosis, *in* F. Kafka, t. W. Muir & E. Muir, eds, 'The Penal Colony', Schocken Books, New York, NY.

Kugel, P. (1986), 'Thinking may be more than computing', *Cognition* **18**, 128–149.

Moor, J. H. (1976), 'An analysis of turing's test', *Philosophical Studies* **30**, 249–257.

Moravec, H. (1999), *Robot: Mere Machine to Transcendant Mind*, Oxford University Press, Oxford, UK.

Pollock, J. (1995), *Cognitive Carpentry: A Blueprint for How to Build a Person*, MIT Press, Cambridge, MA.

Putnam, H. (1994), 'Trial and error predicates and a solution to a problem of mostowski', *Journal of Symbolic Logic* **30**(1), 49–57.

Russell, B. (1936), 'The limits of empiricism', *Proceedings of the Aristotelian Society* **36**, 131–150.

Russell, S. & Norvig, P. (1994), *Artificial Intelligence: A Modern Approach*, Prentice Hall, Saddle River, NJ.

Searle, J. (1980), 'Minds, brains and programs', *Behavioral and Brain Sciences* **3**, 417–424. This paper is available online at http://members.aol.com/NeoNoetics/MindsBrainsPrograms.html.

Siegelmann, H. (1995), 'Computation beyond the turing limit', *Science* **268**, 545–548.

Siegelmann, H. & Sontag, E. (1994), 'Analog computation via neural nets', *Theoretical Computer Science* **131**, 331–360.

Turing, A. (1964), Computing machinery and intelligence, *in* A. R. Anderson, ed., 'Minds and Machines', Prentice-Hall, Englewood Cliffs, NJ, pp. 4–30.

Weyl, H. (1949), *Philosophy of Mathematics and Natural Science*, Princeton University Press, Princeton, NJ.

Wos, L. (1996), *The Automation of Reasoning: An Experimenter's Notebook with* OTTER *Tutorial*, Academic Press, San Diego, CA.