

# On The Claim That A Table-Lookup Program Could Pass The Turing Test

Drew McDermott  
Yale University  
`drew.mcdermott@yale.edu`

July 11, 2013

## Abstract

The claim has often been made that passing the Turing Test would not be sufficient to prove that a computer program was intelligent because a trivial program could do it, namely, a program (the Humongous Table Program, or HTP) that simply looked up what to say next in a table. This claim is examined in detail. Three ground rules are argued for: 1. That the HTP must be exhaustive, and not be based on some vaguely imagined set of tricks. 2. That the HTP must not be created by some set of sentient beings enacting all possible responses. 3. That in the current state of cognitive science it must be an open possibility that a computational model of the human mind will be developed that accounts for at least its nonphenomenological properties. Given ground rule 3, the HTP could simply be an “optimized” version of some computational model of a mind; creating it does not require the creative participation of sentient beings, but just the automatic application of program-transformation rules. Therefore, whatever mental states one would be willing to impute to an ordinary computational model of the human psyche one should be willing to grant to the optimized version as well. Hence it is impossible to dismiss out of hand the possibility that the HTP is intelligent, and the argument loses any force it may have. This conclusion is important because the Humongous-Table-Program Argument is the only argument ever marshalled against the sufficiency of the Turing Test, if we exclude arguments that cognitive science is simply not possible.

## 1 Introduction

This paper is about a fairly narrow claim, which can be phrased crudely thus: Because there is an allegedly trivial program that could pass the Turing Test (henceforth, TT), the Test<sup>1</sup>

---

<sup>1</sup>I will capitalize the word “test” when referring to the Turing Test as a concept, and use lower case when referring to particular test occurrences.

is pointless. The supposedly trivial program in question is one that simply looks up its next contribution to the conversation in a table. It is less clear what more specific adjective the word “pointless” is actually a stand-in for. That depends on what the point of the TT is supposed to be. But in general the thought-experiment is meant to show that seemingly intelligent behaviour can be produced by unintelligent means, whereas real intelligence cannot, so a behavioral test could always be fooled.

My purpose is to argue that the distinction between trivial and nontrivial mechanisms for producing intelligence cannot be so easily drawn. In particular, the conclusion that a table-lookup program is not really intelligent may be due to misplaced focus on the lookup program as if it were the proper point of comparison to a person, instead of considering the table as a whole. When attention is no longer paid to the homunculus, a seemingly unintelligent program might turn out to be an optimized version of an intelligent program, with the same intelligence, just a shorter lifespan. *The allegedly trivial program is in fact not trivial at all.* It could be (for all the argument shows) computationally equivalent to a very clever program indeed.

I hope my argument is not seen as an attempt to shore up the plausibility of the TT, whose problems have been enumerated by many others (most notably Hayes and Ford 1995). But the spell cast by the possibility of the humongous<sup>2</sup>-table program has been simply amazing, considering how little thought has been given to how intelligent such a program would actually be if it existed. In addition, it is the *only* argument I know for the insufficiency of the TT as a demonstration of intelligence, except for arguments that purport to show that cognitive science is simply impossible.

It is a tragicomic sidelight on the tragedy of Alan Turing’s life that he died before the philosophical uproar over his 1950 paper “Computing machinery and intelligence” had really begun. He surely would have clarified exactly what the Test was, and probably would have repudiated many of the philosophical positions attributed to him. As it is, we often have to work out these clarifications for ourselves every time we approach the subject, and this paper is no exception.

So: I will take a Turing test to involve a time-limited exchange of purely textual information between a person called the *judge* or *interrogator* and an entity *E*, called the *interlocutor*, that may be a real human being (the *confederate*) or a computer program (the *examinee*). If after an hour<sup>3</sup> the judge cannot guess with success probability better than a flipped coin which interlocutor is the real human, then the examinee passes the test. Because there is no way to measure a single judge’s success probability, any Turing test would have to be one of a series

---

<sup>2</sup>I realize that the use of this slang term makes the paper sound a bit frivolous. I take this risk because the size of the required table will easily seem to be beyond comprehension, and it’s important to keep this in mind. I don’t think words like “vast,” “stupendous,” “gigantic” really do the job. In (Dennett 1995, ch. 1) the word “Vast” with a capital “v” is used for numbers in the range I discuss in this paper, numbers of magnitude  $10^{100}$  and up.

<sup>3</sup>Or some other arbitrary time limit fixed in advance; but I’ll use an hour as the limit throughout this paper. The importance of this stipulation will be seen below.

of interviews of the examinee and a confederate, each interview involving a different judge, but I will not pay attention to this aspect, or to the intricacies of statistics.<sup>4</sup>

The question is what does the TT test *for*? Turing himself, reflecting the behaviorism of his time, proposed to *replace* the question “Can a machine think?” with the question “Can it pass the Test?” This has been taken as an analysis of the concept of thinking, from which would follow:  $X$  can think if and only if  $X$  can pass the TT. It’s hard to believe that anyone would argue for this definition, and certainly not Alan Turing. As an eager experimentalist in artificial intelligence (AI), who came up with one of the first designs for a chess program based on an idea by Claude Shannon (Hodges 1983; Millican and Clark 1996), he would have surely believed there were programs that acted intelligently but without the capacity for speech. As he points out on p. 435 of (Turing 1950), the Test is meant to be a sufficient condition for intelligence, not a necessary one. In a BBC Radio broadcast (Braithwaite et al. 1952) he described the Test in slightly different terms, and went on to say

My suggestions is just that this [i.e., can a machines pass the Test?] is the question we should discuss. It’s not the same as ‘Do machines think,’ but it seems near enough for our present purposes, and raises much the same difficulties. (Braithwaite et al. 1952, p. 498)

The table-lookup-program argument purporting to show that passing the TT is not even *sufficient* for classing an entity as intelligent was first mentioned casually by John McCarthy (1956) but the first thorough analysis was that of Ned Block (1981). Here is his version: Imagine all possible one-hour conversations between the judge and the program. Of those, select the subset in which the program comes across as intelligent, in the sense of sounding like a perfectly ordinary person. Imagine the utterances of the two participants concatenated together (with the special marker I’ll write as ## between the utterances of one person and the utterances of the other). The result is a “sensible string,” so called because the robot comes across as sensible rather than as, say, an automaton that generates random characters as responses.

Imagine the set of sensible strings recorded on tape and deployed by a very simple machine as follows. The interrogator types in sentence A. The machine

---

<sup>4</sup>Further clerical details: Turns end when the person enters two newlines in a row, or exceeds time or character limits (including further constraints imposed later). As soon as an interrogator tries to type a character that would cause any limit to be exceeded, the chat system types two newlines (to indicate that it’s the interlocutor’s turn) and ignores any characters the interrogator managed to type past the limit. The two newlines between turns don’t count as part of the utterance on either side. We’ll always let the interrogator go first, but they can type the empty string to force the interlocutor to be the first to “speak.” (I will use third-person plural pronouns to refer to a singular person of unimportant, unknown, or generic gender, to avoid having to say “him or her” repeatedly.) The interview ends after an hour or if the interrogator and interlocutor successively type the empty string (in either order). Note that I’ll use sometimes words like “speak” or “say” when I mean “type,” only because the latter sounds awkward in some contexts.

searches its list of sensible strings, picking out those that begin with A[##]. It then picks one of these A-initial strings at random, and types out its second sentence, call it “B.” The interrogator types in sentence C. The machine searches its list, isolating the strings that start with A followed by B followed by C. It picks one of these A[##]B[##]C[##]-initial strings<sup>5</sup> and types out its fourth sentence, and so on (p. 20).<sup>6</sup>

Actually, it’s not necessary for the tape, or database as we would now say, to contain *all* sensible strings beginning with A## and Block himself retracts that idea when he describes a variant of his original machine that represents the TT situation as a game tree. I’ll describe that variant in section 2. I’ll use the phrase “Humongous-Table Program” (HTP) to refer to the Block algorithm, and introduce names for its variants as they come up. Sometimes I’ll use the phrase “Humongous-Table Algorithm” when speaking about it in a more abstract way.

Having defined “intelligence” as “possess[ing] . . . thought or reason” (p. 8), Block concludes

[T]he machine has the intelligence of a toaster. *All the intelligence it exhibits is that of its programmers. . . .* I conclude that the capacity to emit sensible responses is not sufficient for intelligence. . . . (p. 21) [emphasis in original].

I will call this the Humongous-Table Argument (HTA). Block takes this argument to refute a behavioristic definition of “conversational intelligence,” defined thus:

Intelligence (or, more accurately, conversational intelligence) is the capacity to produce a sensible sequence of verbal responses to a sequence of verbal stimuli, whatever they may be.

Block proposes an alternative framework, which he calls *psychologism*, for understanding the mind. He defines this as “the doctrine that whether behavior is intelligent behavior depends on the character of the internal information processing that produces it” (p. 5).<sup>7</sup> It is fair to say that almost everyone who hears the HTA agrees with its conclusion.

---

<sup>5</sup>It’s obviously necessary to insert something like the ## marks because otherwise there would be many possible interchanges that could begin ABC. (How would you separate “Veni . . . Vidi . . . Vici. Ave Caesar! A fellow Latin scholar. Great!” into three components?) Block probably just assumed some such marker would end A, B, and C. I’m making it explicit.

<sup>6</sup>Actually, just to get the chronology right, it’s important to note that Block described a slightly different version of the program in (Block 1978, p. 281) in order to make a somewhat different point. Very confusingly, an anthology published two years later, included a slightly condensed version of the paper under the same title (Block 1980), a version that lacks any mention of the humongous-table program.

<sup>7</sup>Shannon and McCarthy 1956 believe that a definition of “thinking,” in the case of an allegedly intelligent machine, “must involve something relating to the manner in which the machine arrives at its responses.”

Unfortunately, it's also fair to say that the suggestions made by Block and others have led many people to draw the conclusion that there are always a bunch of "tricks" for faking intelligence available to wily programmers. Hence the ability to pass the Turing Test *provides no evidence at all* that an entity is intelligent. I hope that by refuting Block's argument I can also pull the rug out from under this intuition.

Let us note here that, for the HTA to work, two aspects of the test are crucial:

1. There must be a fixed time limit for the conversation between judge and interlocutor. Otherwise a simple table wouldn't be adequate.
2. The judges must not be able to compare notes before deciding whether an interlocutor is human. Otherwise the fact that the same table governs all the conversations might give the game away.

In section 2 I will talk about further developments and amplifications of the humongous-table argument. In section 3 I will propose some ground rules for thinking about the argument. In section 4 I will lay out a counterargument that refutes the HTA. In section 5 I will summarize my conclusions.

I apologize for the length of this paper. Block took a long time setting the HTA up, and it will take me a while to dismantle it.

## 2 The Argument and Its Role

The humongous-table argument has been frequently cited or independently discovered, and almost always believed. A typical allusion in the AI literature is from (Perlis 2005):

What might such evidence be [that, despite passing the Test, a system is not intelligent]? One answer is obvious: the system turns out to have an enormous database of answers to everything, and simply performs a lookup whenever called upon to respond.

And here's one from the philosophical literature. Robert Kirk, after describing Block's program, comments

All the program has to do is record whatever has been typed so far, match it against one of the stored conversations, and put out the next contribution ... in

that stored dialogue. Such matching is entirely automatic, and clearly calls for no intelligence on the part of the program. Nor does the system work out its own responses: that was done in advance by those who prepared the possible dialogues. So although Block's machine produces sensible responses, it is not intelligent. (Kirk 1995, p. 398)

Copeland and Proudfoot (2009) seem to assume that the HTA is correct, but claim that because the lookup-table is too big to exist or be searched in the real world, the conclusion matters only if Turing intended to define "think,"

... because, in that case, Turing would indeed have had to say that 'If machine  $M$  does well in the imitation game, then  $M$  thinks' is true in *all* possible worlds. ... At bottom, then, the [HTA] depends on the interpretive mistake of taking Turing to have proposed a definition. ... There is no textual evidence to suggest that Turing was claiming anything more than that 'If machine  $M$  does well in the imitation game then  $M$  thinks' is *actually* true, that is, true in the actual world. Nor did Turing need to claim more than this in order to advocate the imitation game as a satisfactory real world test. (p. 131)

I agree that Turing did not intend to define "think" (cf. (Dennett 1985)), but I confess I don't see what difference the possible worlds make.

Dowe and Hajek (1997,1998) make a proposal for additional requirements on any TT examinee that seems to take the HTA for granted. These requirements amount to demanding that a program be of reasonable size, so that it achieves "compression" of information about the world; the HTP wouldn't satisfy this demand.

Braddon-Mitchell and Jackson, in a recent textbook (Braddon-Mitchell and Jackson 2007) on philosophy of mind, use Block's argument to dispose of what they call *input-output functionalism*, defined as the thesis that any entity with the right kind of behavior (inputs and outputs) would have mental states — "a mind" (p. 111f).

Braddon-Mitchell and Jackson describe the algorithm as if it were playing a game. The "variant" algorithm of (Block 1981, p. 20) is quite similar, and we'll stick with Block's version for reasons that I'll explain later. We can represent what the robot needs to know using a tree of the sort depicted in figure 1. The interrogator goes first. They might type any paragraph below some set length (say, 500 characters), such as "It's so sunny today." For each such input, "the programmers produce *one* sensible response" (Block 1981, p. 20) (my emphasis). In figure 1, we move from the *root node* (a *J-node*, colored white) down the *edge* labeled with the string "It's so sunny today". The resulting node (a *C-node*, colored black) has just one edge leading out

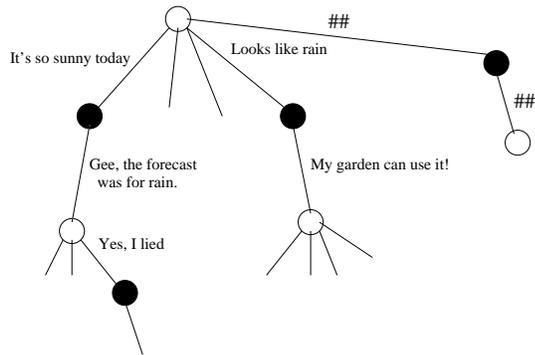


Figure 1: A few nodes of a strategy tree. White circles represent *J-nodes* where the judge is to speak; black circles represent *C-nodes* where it's the computer's turn.

of it, labeled with the chosen sensible response: "Gee, the forecast was for rain." Moving down that edge, we reach another J-node, where the program must once again be prepared for *any* character string by the interrogator; but after hearing that string, there can again be a single designated response.<sup>8</sup>

I will use the term *strategy tree* for a tree as described in figure 1, for reasons that will become clear. Having described the HTP as a strategy tree, Braddon-Mitchell and Jackson then agree with Block that the "clearest intuition" about it is that "it completely lacks intelligence and understanding" (p. 115).

We now have two versions of the HTP on the table:

HTPL: As a long list of "sensible strings" and a simple program to search it.

HTPS: As a strategy tree.

To see that these are equivalent, let's be more specific about the first case. Remember that we introduced a special marker ## to separate utterances (not allowed within them). So the program to search the sensible-string list looking for (the) one beginning  $A##B##C##$  can be thought of as searching through a table in which the keys are strings with an even number of occurrences of ##<sup>9</sup> and the values are strings with no occurrences of ##. The table might contain the following key-value pair:

<sup>8</sup>Block talks as though the "programmers" might emulate his Aunt Bertha. Actually, they can be somewhat more creative if they want to. On different branches of the tree, different "personalities" might emerge. But it will be much simpler, and sacrifice no generality, to speak as though each tree emulated one personality, and we'll go along with calling her "Aunt Bertha" or "AB." I have my doubts that we will ever be able to simulate a particular person in enough detail to fool their close friends. But that's not necessary. If someone creates a program to compete in a Turing test and bases it on their aunt, it doesn't have to mimic her that closely. If it sounds to the judges like it might be someone's aunt, that's good enough.

<sup>9</sup>Equivalently, odd-length *lists* of strings.

It's so sunny today##Gee, the forecast was for rain.##Yes, I lied.

→

I guess I pass the first part of the test!

meaning that if the conversation has so far been:

*Judge:* It's so sunny today

*Interlocutor:* Gee, the forecast was for rain.

*Judge:* Yes, I lied.

the computer should next utter

I guess I pass the first part of the test!

The relationship between HTPL and HTPS is that a key (string separated by ##'s) in HTPL corresponds to the labels on the edges leading from the top J-node down to the current C-node. For example, in figure 1, the C-node on the bottom left is reachable through the three edges from the top node that are labeled with the strings leading to it in the game. There is a one-to-one correspondence between C-nodes and odd-length tuples of strings. However, the HTPS gives us more degrees of freedom, because the nodes of the tree are *abstract*: they can be realized physically in many different ways, provided the realization supports the state transitions shown in figure 1. (We'll pick up this thread again in section 4.1.) In what follows I will describe the HTP as HTPL or HTPS, whichever is convenient.

Actually, figure 1 suggests an optimization of HTPL: because the examinee's responses are completely determined by the judge's inputs so far, we don't need to include them in the keys to the table. The key-value pair used above can be shortened to

It's so sunny today##Yes, I lied.

→

I guess I pass the first part of the test!

because the middle segment of the key ("Gee, the forecast was for rain.") never varies given the first segment. From now on, therefore, we will assume that key strings are of length  $k$ , if the judge has typed  $k$  questions so far, not  $2k - 1$ .

The responses at C-nodes can't be simple strings, because of the timing issue. Suppose we give Aunt Bertha two problems:

What's 347 + 702?  
Divide 40803 by 203

Assuming that arithmetic doesn't just give her the vapors, she would have to pause to figure these out, and it's likely that she would take longer on the second problem than on the first. If the strings are printed out at some fixed rate, or if random time delays are introduced between characters, the judge will easily unmask the program. One solution is to imagine that each output string is accompanied by a list of monotonically increasing times in milliseconds, where the list is as long as the string. Given these two lists, one of characters, the other of times, the HTP will start the clock as soon as it receives the input, then print character 1 at time 1, character 2 at time 2, and so forth.<sup>10</sup> This will allow a delay at the beginning for Aunt Bertha to think her answer over, plus delays while she thinks or mulls her choice of words. In what follows, to minimize clutter I am going to ignore these timings most of the time, but they must be present in all variants of the HTP that I will discuss. I will use the term *timed string* to refer to these  $\langle \text{string}, \text{timing-data} \rangle$  pairs.<sup>11</sup> I will leave modification of Block's sensible-string table to include timing data as an exercise.<sup>12</sup>

I'll assume that timed strings are generated and then some interface program handles the waiting for the prescribed times, and that the strings are generated fast enough that the deadlines can be met for each character to be typed.

I've introduced the restriction that the judge can type no more than 500 characters. We could allow sentences of arbitrary length, so long as they could be typed in the time remaining. So initially the allowed strings could be longer, and get shorter near the end. It's simpler, and doesn't contradict the spirit of Turing's proposal, to set a fixed length, somewhat more generous than Twitter's. Because neither judges, confederates, nor examinees type at unbounded speeds, the tree of figure 1 is finite. A *leaf* is a node with no children, for one of two reasons: both parties have just typed the empty string in succession; or it would take at least an hour to reach the node given the maximum speed judges can type. The only leaf node visible in figure 1 is the J-node on the far right. The transitions marked with ## indicate that the empty string was typed. Apparently the examinee in figure 1 refuses to proceed with the interview if the judge does not type something to open the conversation. We'll define a *branch* as a sequence of nodes from the root to a leaf.

---

<sup>10</sup>No time can be greater than the number of milliseconds in an hour, but at "run time" the actual time left determines whether the interview comes to an end before the judge and examinee give the signal.

<sup>11</sup>If we want to allow interlocutors to edit lines before they are seen by the interrogator, then times should be associated with completed lines, not individual characters. If we really want to avoid reaction times completely, then we could have two sets of judges, one to conduct the interviews and another to review the transcripts and decide who's human. But that's a rather drastic change to the rules.

<sup>12</sup>One more restriction: Timed strings can't have times so short that the typing speed exceeds the rate a plausible human can type. Of course, if the examinee types at blinding speed it will be easy for the judge to identify, but if we're considering the space of all possible examinees, as we will in section 4.2, it's necessary to set bounds on their abilities to keep the table finite.

What number is this:

```
////_xx_____xx
////_xx_____xx
////_xx_____xx
////_xxxxxxxxxx
////_____xx
////_____xx
////_____xx
?
```

Figure 2: An input we shouldn't require an examinee to handle.

It is still possible within the 500-character limit for the judge to type inputs such as the one in figure 2. We'd like to rule out such "pictorial" concoctions. One way is to convert newlines ("carriage returns") to single spaces, but nothing would forbid the judge from asking, "What number is this when each sequence of slashes is converted to a newline:..."? A better idea is to restrict the number of characters that can occur outside of words. I'll explore some ideas along these lines below (section 3.1), although I grant that a determined judge can defeat all such rules by using several turns to type their question in.

There is another issue that I feel queasy about, and that's how to deal with timing the *judge's* inputs. So far there's nothing to prevent the judge from typing

```
I'm going to type the same sentence twice, once slowly, once quickly:
Mary had a little lamb.
Mary had a little lamb.
Which was which?
```

Including timing data in the examinee's outputs is a nuisance. But including them in the *interrogator's* inputs causes a mind-boggling increase in combinatorics. The table must anticipate every possible input, which means not just enumerating all possible strings, but all possible *string timings*. Even Block might blanch at the thought.

I am going to assume that the judge's input is collected in its entirety and then fed to the interlocutor as a chunk, so that this issue can be ignored.

A strategy tree, such as that of figure 1, is not a game tree in the familiar sense (Russell and Norvig 2010, ch. 5). There is no lookahead depth, static evaluation, or any of that. It is essentially what game theorists call a *pure strategy in extensive form* (Binmore 2007) for playing the computer's side of the Turing Test game. It lays out exactly what to do in every

situation that can occur during the game. Nodes with an edge for every string the “opponent” might utter alternate with nodes containing just one edge, the machine’s response. The only thing missing is the payoff at the end, which is, unlike in game theory, unpredictable, being set by the interrogator after the game has been played. Of course, the “programmers,” if they’ve done their job right, will have caused one end of a plausible conversation to be typed out by the time any terminal node is reached.<sup>13</sup>

In addition to passing the Turing Test, both Block and Braddon-Mitchell and Jackson believe the Humongous-Table Program can be adapted to control *all* the behavior of a person.

The machine, as I have described it thus far, is limited to typewritten inputs and outputs. *But this limitation is inessential, and that is what makes my argument relevant to a full-blooded behaviorist theory of intelligence, not just to a theory of conversational intelligence* (Block 1981, p. 23) [emphasis in original].

Braddon-Mitchell and Jackson, in fact, put their entire discussion in terms of an automaton for playing “the game of life,” that is, having *all* its behavior governed by a version of the HTP.

... [A]t any point in a creature’s life there are only finitely many discriminably distinct possible inputs and outputs; but in any case we know that there is a limit to how finely we distinguish different impacts on our surfaces and to how many different movements and responses our bodies can make. This means that in principle there could be a ‘game-of-life’ [strategy] tree written for any one of us . . . .

We list all the different possible combinations of pressure, light, gravity and so on impacting on the surface of [Aunt Bertha’s] body at the first moment of [her] life.<sup>14</sup> [These would be the edge labels below the top node] of the game-of-life [strategy] tree modelled on [Bertha]. . . . [The edge labels for the edges coming out of all the resulting C-nodes] would give the behavioral response that [Bertha] would make . . . to each possible input. (Braddon-Mitchell and Jackson 2007, p. 114) [Sorry for all the brackets; I’m translating into my terminology.]

The “behavioral responses” are quanta of energy pumped into muscles over the next time interval. Getting this discretization right is not easy at all, and is one of the major issues in digital control theory (Leigh 2006).

A humanoid robot that can fool a person into thinking it’s human passes what Harnad 1991 calls the Total Turing Test (TTT):

---

<sup>13</sup>For now, I will be casual about the distinction between a strategy tree — a mathematical object — and a representation of a strategy tree in a physical medium. How the latter might work is discussed in section 4.1.

<sup>14</sup>Braddon-Mitchell and Jackson seem oddly oblivious to the fact that real people grow and then wither over their lifespans. Perhaps “behavior” for them includes changes in body shape. For our purposes the robot’s lifespan need merely be an hour.

“The candidate [automaton] must be able to do, in the real world of objects and people, everything that real people can do, in a way that is indistinguishable (to a person) from the way real people do it” (Harnad 1991, p. 44).<sup>15</sup>

Adapting the HTP to pass the TTT is essentially what Braddon-Mitchell and Jackson are proposing. However, I am going to neglect the TTT, and focus on the plain-vanilla Turing Test, for several reasons:

1. If there is reason to believe that the Humongous-Table Algorithm has mental states in Turing’s scenario, then it presumably does in more complex scenarios as well.
2. In case it’s not clear already, I hope to persuade you that the HTP is truly humongous, even in the ordinary Turing scenario, but the HTP that would be required to control a robot for an hour or more is many, many orders of magnitude larger.
3. Any branch through the tree must be physically possible. But that means the inputs to the tree reflect a ghostly model of the physics of the universe in Aunt Bertha’s vicinity.<sup>16</sup> As far as “Aunt Bertha” is concerned, it is as if there is an evil deceiver producing a physically accurate universe — no, all possible physically accurate universes. But what’s physically possible depends on unknowable boundary conditions. If a jet flies overhead, the noise might require comment. Do we have to think about objects as far away as jet airplanes? And what about the behavior of other intelligent agents in the vicinity? If the interrogator challenges the robot to a game of ping-pong, the model of the interrogator’s motions must combine physics with game theory. We may end up enumerating the possible positions and momenta of all atoms within a sphere with radius one light-hour. We probably don’t have to descend to the level of elementary particles, but who knows?
4. The rules of the TTT are even less well spelled out than the original. (I think it’s fair to say that the details of how to run these tests do not interest Harnad, the TTT’s original proposer.) Where are the judges? Who knows that they *are* judges? (Perhaps it’s like a mixer: We collect a bunch of real people and humanoid robots in a room, and encourage them to mingle; the people aren’t told who is real. At the end of the evening the people are asked to rate their fellow participants on a humanness scale.)
5. We have to settle the question: Do the robots know they’re in a contest? Or do they believe they’re people?

---

<sup>15</sup>This test is a blend of what call Harnad calls T3 and T4 in (Harnad 2000), depending on whether the automaton has to be able to do things like blush or not.

<sup>16</sup>If we opt instead for all *mathematically* possible input sequences, then for all but a vanishingly small fraction scientific induction does not work; the universe is mostly white noise. In the ones where scientific induction does work, all but a vanishingly small fraction have different laws of nature from those in the real world. At this point I no longer believe that the game tree has been specified precisely enough for me to conceive of it.

This last question may seem odd, but in the TT it seems you would answer it one way, and in the TTT the other. In the TT, the best strategy for the computer is to act human in every way. So there is no need for a layer of “personness” that believes it is really a program taking part in a test. If you ask an Aunt Bertha simulator whether it’s a program, it might respond, “My goodness! Won’t my husband get a chuckle out of that idea!” It is not as if somewhere in its database is represented the belief that it is lying.<sup>17</sup> But the robots in the TTT must know they’re pretending to be human. Otherwise they might wonder why their arms seem to weigh a bit more than other people’s; where their tough skin came from; when they’re going to get thirsty or hungry; where their families are. A judge could do better than chance by check out who goes to the bathroom and who doesn’t. A daring judge could see who responds to flirting and how far they seem willing to go. A robot would have to deliberately lie in these situations. For example it might pretend to use the bathroom.

I will avoid all these complexities by focusing on the plain-vanilla TT from now on. And I will assume a “delusional” examinee, that thinks it really is Aunt Bertha sitting in front of a computer.

### 3 Some Ground Rules

In this section I will argue for three premises that are necessary to make it clear just how hard it would be to build the Humongous-Table Program, and what resources are available to build it.

#### 3.1 Only Truly Exhaustive Programs Are Allowed

Our first ground rule is that the Humongous-Table Algorithm must be what Block first proposed: An exhaustive representation of a strategy tree. As soon as we introduce little tricks or shortcuts, we’re doing (bad) AI research. There’s no plausible short list of tricks that will make the Humongous-Table Program significantly smaller while preserving its ability to pass the TT *and* its apparent triviality. One is not allowed to list a few tricks and then say, “Just add a few more and you could probably fool the average interrogator.” In fact, in spite of some early success with naive subjects, reportedly anecdotally by (Weizenbaum 1976) and (Humphrys

---

<sup>17</sup>Of course, a truly intelligent chatbot would have to have delusional beliefs about its physical appearance, so as to be able to answer questions such as “How tall are you, Bertha?”, and “Are you left- or right-handed?” (And about its surroundings; see section 3.3.) It will also have to have delusional memories of, say, having eaten strawberries and cream, or having ridden a snowmobile, or having done *some* real-world thing, or the judges will get suspicious. Whether we can attribute even delusional beliefs to the HTP is an issue we take up in section 3.3.

Strategically optimal or not, is it ethical to create a model of a person, run it for an hour so it can take a test, reset it to its original state, run it again a few times, then junk it?

2008), among others, no program has come close to passing the Turing Test (Christian 2011). The annual Loebner Prize competition, held ostensibly to conduct a Turing test, imposes rules that make a serious test impossible: Interviews are short (5 minutes, usually), the judges are naive about the Test and about the state of AI, and they are even asked not to probe the contestants too sharply. Some of the papers by recent contestants in (Epstein et al. 2008) are disappointingly cynical about how to do well: create zany characters, ask the judges questions so they'll talk about themselves, and print out long, intricate, canned outputs to eat up the clock. The resulting programs are nothing like the Humongous-Table Program, except that most of them involve scripts that give ways of reacting to typical inputs. These scripts are small, and use matching and registers (see below) to be able to react more flexibly with fewer rules than the HTP. The patterns they use to match sentences take no account of syntax. Rules may be used multiple times per conversation. The programs have no ability to reason, no knowledge of anything, and no ability to grasp the topic or maintain the thread of the conversation. They are ridiculously easy to expose.

Ground Rule 1 forbids all such trickery. Here's an example of what the rule forbids. Suppose the judge says early on,

“My name is Pradeep. Remember that, because I'm going to ask you about it later.”

In any subtree beginning with this utterance, whenever the judge asks, “What's my name?,” the program should probably say, “Pradeep.” But the judge might have substituted “Fred,” or “Michio,” or any of a huge number of other names. All of the subtrees rooted after these alternative utterances could be identical except for having to substitute “Michio” or “Fred” for “Pradeep.” The temptation is great to avail yourself of a simple device to record that information, so we need to have only one tree, to be used after any utterance of the form

“My name is ?R. Remember that, because I'm going to ask you about it later.”

The variable ?R matches “Pradeep” or “Michio” or whatever, and the name is placed in a storage location labeled ?R. These locations are called *registers*. On occasions where the name should be recalled, we substitute the contents of register ?R instead: “I haven't forgotten: It's ?R.” Now all the trees differing only in which name is remembered can be collapsed back into one tree.

Around p. 33 of (Block 1981), Block begins to indulge in just this sort of “hacking.” Suppose we *limit the size of the vocabulary*; and we *allow the Aunt Bertha simulation to get confused*; or *have Aunt Bertha blather on* without regard to the input. (The last two suggestions allow parts of the strategy tree to be shared, which amounts to “forgetting” how the possible paths to the shared subtree differ.)

If one sets one's sights on making a machine that does only as well in the Turing Test as most people would do, one might try a hybrid machine, containing a relatively small number of trees plus a bag of tricks of the sort used in Weizenbaum's [ELIZA] program.

He speculates that the HTP might be “nomologically possible” after all:

Perhaps many tricks can be found to reduce the memory load . . . . [N]o matter how sophisticated the memory-reduction tricks we find, they merely postpone, but do not avoid the inevitable combinatorial explosion. . . . [T]echnical ingenuity being what it is, the point at which nomological impossibility sets in may be beyond what is required to simulate human conversational abilities (Block 1981, p. 34).

But it must be kept in mind that one man's tricks are another man's trade. If by “tricks” is meant “programming techniques generally,” then Block seems to be suggesting that any program that passed the Turing Test by being an ordinary person (and not a genius) could be construed as the application of some tricks to an exhaustive program, thus cutting it down to size; and I fear that this is how he has been read (although I don't think that's what he intended to say). If someone could really simplify the Humongous-Table Program using registers and some of these other tricks, in such a way that it still resembled an exhaustive strategy tree, and could still pass the Turing Test, then we would have a different argument to contend with. So let's try to estimate just how big a strategy tree they are chipping away at with the standard tricks.

It is not hard to see that the size of *just one node* of a strategy tree used by the HTP is so huge that it will be hard to keep it in the known universe. Let's see what we can do to make the tree smaller. In figure 1 it is assumed that for every string typable by the judge, there is an edge descending from each J-node labeled with that string. There are 127 Ascii characters, but many of them are control characters. There are actually only 94 “visible” characters, plus space. (I proposed above that newlines and tabs be converted to spaces to make it hard to draw pictures with characters.) So we have to worry about “only”  $95^{500} \approx 7.3 \times 10^{988}$  ASCII strings of length 500. But all but a tiny fraction of these strings are complete gibberish.<sup>18</sup> Let's try to cut down the number by requiring the judge to type a word string with a few additions. To be precise, suppose we require the “backbone” of an input to be a string of words from a dictionary of the 10,000 most common words, with a few additions:

1. The judge can make up some new words (e.g., names), but may use no more than 40 characters per input for this purpose. The new words are interleaved arbitrarily with the string of common words, for a total of no more than 85 words.

---

<sup>18</sup>Jorge Luis Borges's vision in “The Library of Babel” conveys the idea.

2. Up to 10 punctuation characters are added, in bunches that must come immediately before or after a word; where a punctuation character is any nonspace character other than a letter or digit.

An extremely crude estimate of the number of inputs these constraints allow is  $3.3 \times 10^{367}$ . This looks like a considerable improvement on  $7.3 \times 10^{988}$ , and I suppose it is, but it is still a number too big to really comprehend. (For comparison, the volume of the observable universe in cubic Planck lengths is about  $2 \times 10^{185}$ .) There is no way that an object with this many pieces will fit in our universe (barring new discoveries in physics); and remember that this is for *one node* of the strategy tree. The entire tree, if it covers roughly 50 exchanges per interview between judge and examinee, will have  $(10^{367})^{50}$ , or about  $10^{17,500}$  nodes. As Block admits, it's hard to picture such a thing existing, unless we discover new laws of physics that allow parts of spacetime to be indefinitely subdivisible.

No one has ever made a realistic proposal for reducing the size of this tree significantly without changing the whole approach, thinking seriously about knowledge representation, the interaction of syntax, semantics, and pragmatics, and how participants in a conversation navigate through topics, in other words, doing serious research in AI and computational linguistics. The burden of proof is squarely on the person that claims they have an “almost exhaustive” program that avoids solving all these difficult research problems and can pass the TT. Until this program appears, we adopt Ground Rule 1: the Humongous-Table Program must remain exhaustive, and include a representation of all the possible sensible pieces of a conversation. Even if you think this rule is overly rigid, it's clear that using registers and similar tricks is not going to make a serious dent in  $10^{17,500}$ , and the rule forces us to bear that in mind.

Sometimes I think when people say that an unintelligent program could pass the Turing Test, what is going through their minds is something like this:

1. They've seen one of the “chatbots” on the Internet, or in transcripts of the Loebner competition.
2. They've heard enough about how these programs work — or how Eliza worked (Weizenbaum 1976) — to know that they're trivial.
3. They vaguely imagine that you could, say, make the one of these program 10 times longer, and it would survive an interview 10 times longer than allowed in the Loebner competition. Or something like that.
4. Hence a trivial program could pass the Turing Test.<sup>19</sup>

---

<sup>19</sup>And people do say this. One reader of a previous draft of this paper, who shall remain nameless, speculated that a “better [chatbot] program” “can pass the TT, for all we know.” I just don't see where this intuition comes from. If we know anything we know that a chatbot cannot pass the Turing Test.

The induction embodied in this “argument” might have seemed plausible to some people in 1976, but it shouldn’t now. Still, people might think that Block’s argument is essentially the same as this one, talking about a program much bigger than today’s chatbot, but hey, computers’ powers do grow exponentially, so surely any year now we’re going to be able to build and store the table Block described, right?

Unfortunately for this argument, we could fill the entire visible universe with the strategy tree and need more space. But one’s ability to conceive of the HTP is about to be strained further.

### 3.2 The Sensible-String Table Must Not Have Been Built By Enacting All Possible Conversations

It is difficult to imagine the strategy tree coming into existence, in the sense of being built by human agency. We started by visualizing a team of programmers, but we now realize that the team would have to be impossibly large, and work for an impossibly long time. Actually, “programmer” is not a good job description here; “actor/scriptwriter” would be better, because what the team would be doing is imagining all  $10^{17,500}$  conversations Aunt Bertha could find herself in. Picture a team of scriptwriters hired to write all possible scripts for a given character, each scriptwriter paired with an actor<sup>20</sup> to work out the timings of the characters; but I’ll use the term *actor* for brevity. Obviously, this is not likely to be nomologically possible. But it *might* be, according to Block:

Suppose there is a part of the universe (possibly this one) in which matter is infinitely divisible. In that part of the universe there need be no upper bound on the amount of information storable in a given finite space. So my machine could perhaps exist, its tapes stored in a volume the size of, e.g., a human head. Indeed, one can imagine that where matter is infinitely divisible, there are creatures of all sizes, including creatures the size of electrons who agree to do the recording for us ... (Block 1981, p. 32).

I will use the term *Unbollywood* for this hypothetical region of hyperspace or infinitely divisible segment of our universe, short for “unbounded Hollywood.”

Alas, although we can *place* (the representation of) the strategy tree in Unbollywood, we cannot *build* it there. The problem is that our little mega-teams of tiny actors presumably have mental states. At this point in the argument we haven’t got any way to produce Aunt-Bertha-style behavior without having actors imagine how AB will react to various conversational situations. Hence the strategy tree is essentially a recording device for the mental states of these actors.

---

<sup>20</sup>And perhaps a cognitive psychologist.

The point of the machine example may be illuminated by comparing it with a two-way radio. If one is speaking to an intelligent person over a two-way radio, the radio will normally emit sensible replies to whatever one says. But the radio does not do this in virtue of a capacity to make sensible replies that it possesses. The two-way radio is like my machine in being a conduit for intelligence, but the two devices differ in that my machine has a crucial capacity that the two-way radio lacks. In my machine, no causal signals from the interrogators reach those who think up the responses, but in the case of the two-way radio, the person who thinks up the responses has to hear the questions. In the case of my machine, the causal efficacy of the programmers is limited to what they have stored in the machine before the interrogator begins. (Block 1981, p. 22)

Braddon-Mitchell and Jackson (2007, p. 112) also explicitly rule out the case of an automaton that is remotely controlled by “puppeteers” via a two-way radio. Assume they and Block are right. Does it matter whether the responses are recorded by the puppeteers in advance, or generated on the fly? Richard Purtill argues that it doesn’t. As he says, in connection with a simplified version of the TT (Purtill 1971, p. 291):

... [I]f we think about programming the game a fundamental difficulty comes to light. Where would the computer’s answers come from? Would they simply be written out by the programmer and stored in the machine? But then, of course, the [interrogator] would be right whichever respondent he chose as the human being. A set of answers will either have been given by the human [confederate], who typed the answers himself . . . , or else they were given by the programmer, also a human being, who transmitted them to the [interrogator] via the computer. The question then becomes really “Which set of human-originated answers were transmitted via computer, which were not?” [Bracketed words indicate substitution of my terminology for Purtill’s.]

So the argument for ground rule 2 begins with this premise: The automaton must be self-contained and not simply be a conduit for the mental states of other beings.

Of course, there have always been those who believe that *any* algorithm can reflect only the mental states of its programmers. But I presume this is nowadays recognized as a fallacy. It is possible to write a program to play a game, or learn to play it, much better than the programmer does. So if this program comes to realize that it can force a win in a particular situation, this belief (if that’s what it is) was never a mental state of its programmer, who could never have seen that deeply into the game. I mention this to scotch the idea that a mental model of the sort to be discussed in section 3.3 would inherently be a mere transmission channel between whoever built it and those who interact with it.

Getting back to the main line of our argument, let me show in detail why we cannot in fact picture the strategy tree coming into existence by the action of actors in Unbollywood. The argument is of a familiar type. We start from a scenario that obviously has feature  $F$ , and gradually change it into a target scenario, crossing no line at which  $F$  would be lost. Let's assume our actors are able to improvise, and let's start by simply having one of them take the TT, pretending to be Aunt Bertha; that is, there is no computer or strategy tree, just a textual-communication line between the judge and the actor. Obviously, as Braddon-Mitchell and Jackson argue, anyone who entered this actor as an examinee would be accused of cheating, or of not understanding the rules.

Earlier we counted  $10^{367}$  possible inputs to the program. In the second scenario in our sequence we hire  $10^{367}$  (sub-electron-size) actors. Each possible input from the judge is assigned to one actor. He or she decides what Aunt Bertha would say if their input were the judge's next query. What they type is recorded (as a timed string). We can view the recordings as a one-layer strategy tree. All but one of these actors was assigned what turned out to be an inaccurate prediction of the next input. The winner's output is played back to the judge. Then all the actors try to predict the next input, and the process repeats. The actors don't have to have the same conception of how Aunt Bertha would behave; but they're all competent to continue the dialogue started by the one who was assigned the correct input prediction. Clearly, this group of  $10^{367}$  people is not a legal examinee either. The sentence is recorded, but it's still being transmitted from the brains of a group of actors to the judge; the recording medium is just a transmission line with a delay.

Actually, we don't have to have  $10^{367}$  actors; we could make do with fewer if each actor handles several edges, recording their response to one input, then going back to think about how they would respond to a different input, and so on. It all depends on how much faster these actors can think in their region of Unbollywood than normal people can.

Now switch to a crowd of  $(10^{367})^2 = 10^{734}$  actors, each of whom is given a two-input sequence to think about. More precisely, we first use  $10^{367}$  actors to record a response for each potential input. Then, we use (about)  $10^{734}$  actors to record, for each such input-and-response, a response to every possible second input. The result is a two-layer strategy tree. Now the judge begins the conversation; his or her first input allows us to discard all but one of the first-level recordings, and allows the actors to begin work on the next layer. The job of the actors is to stay two layers ahead of the judge before getting the judge's next input. As before, if the actors were very speedy, we could make do with fewer of them. If they thought  $10^{367}$  times as fast as a real person, we would need only  $10^{367}$  of them. But however we do it, this improv troupe is still not a legitimate competitor; in the end, the judge is talking to a set of sentient actors, with a delay of two layers. Next we find  $(10^{367})^3 = 10^{1101}$  actors, or fewer if they're speedy, and build a three-layer strategy tree. As the judge's actual inputs come in, the actors stay three layers ahead.

If we continue this process, we will eventually reach a crowd of somewhere around  $10^{17,500}$  actors recording the entire strategy tree in advance, then waiting to see which of its branches actually occurs. That one is played back to the judge, just as for the smaller trees. If none of the intermediate trees are legitimate examinees in the TT setup, then neither is the final tree.

One might object that in my scenario the tree is built in “real time.” That is, the delay between the recording of the actor’s responses and the playback is short, no more than an hour, if that’s the maximum duration of the conversation. Just increase the delay and eventually you’re no longer communicating with the actors in any sense. But there is no credible boundary, even a fuzzy one, between sufficient delays and insufficient delays.

This argument establishes ground rule 2, that disqualifies as a legal examinee any program consisting of a strategy tree that came into existence as the result of the work of a team of actors, even though that was Block’s and everybody else’s picture of how it would be created. No other way of constructing the exhaustive strategy tree has ever been proposed.

Another way we can allow the object encoding the tree to come into being is spontaneously, perhaps as part of the Big Bang. (Invoking such events is a standard move in philosophy arguments, a move that has brought us such entities as Swampman, an exact duplicate of, say, Donald Davidson, who just happens to be formed by random processes at work in a swamp (Davidson 1987).) The object must come with an I/O interface we can hook up to our computers, plus some kind of user’s guide and a guarantee (signed by God?) that it actually contains a static strategy tree and is not in reality a hive of electron-size beings who are pretending to be Aunt Bertha in real time, or with a delay. There is no way we could ever *verify* any of this, because even with tools allowing us to inspect the infinitely divisible region of space containing the object it would be impossible for the human race to inspect more than an infinitesimal fragment of the strategy tree in the lifetime of the universe. But we are talking about a conceptual possibility, not about epistemology. We imagine that a robot controlled by an HTP rolls into the Turing-test venue and asks to compete. It’s irrelevant that if it passes no one is likely ever to *believe* that it is controlled by an HTP.

If we start imagining that a model of Aunt Bertha could come into existence spontaneously, then we must ponder the other tables that could come into existence. There is a table that answers all yes/no mathematical questions. That is, the judge types in a conjecture, and the table prints out a proof that it is true, or a counterexample (or a proof that there is one). If it takes longer than an hour to print all this out at top speed, no problem. Just write the conjecture followed by the phrase “Hour 2” and the machine will start typing the second hour’s worth of proof or counterexample. How can a finite object solve an infinite number of problems? It can’t, quite. If the conjecture followed by the hour number take more than one hour to type then we are stuck. But the table would settle all conjectures of interest to the human race.

Another table in this space solves all instances of the Halting Problem (Homer and Selman 2011, ch. 3) of any reasonable size, although there would be no way to know that it did unless

it provided some kind of proof of its answers, which would revert to the previous case.

There is also a table that composes symphonies in the style of Mozart or Beethoven, or both (just type the composer you want). We are in Borges territory here.<sup>21</sup> Although the set of all possible tables includes tables that perform amazing feats of creation,<sup>22</sup> all but an infinitesimal fraction of them just print gibberish, carefully but bizarrely timed gibberish.

There is, however, one advantage to assuming a random or miraculous process: it allows us to solve the “daily context” problem. Suppose the judge starts talking about current affairs with a program whose development began long enough ago for a programming team with fewer than  $10^{17,500}$  members to have enacted all possible conversations with its target character, Aunt Bertha. “Long enough ago” may be thousands or millions of years, so “Aunt Bertha” may know nothing about recent events such as happenings in the news, or the evolution of the English language, or the development of agriculture, or the emergence of multicelled life.<sup>23</sup> We solve that problem by imagining that the strategy tree just happens to be equipped with up-to-the minute information of events recent at the time the Turing test is run.

If all of this seems preposterously unlikely, good. In section 4, I provide a much more satisfying way the HTP can come into existence, without the use of sentient actors.

### 3.3 If We Neglect Phenomenology, Computational Models Of People Are Possible

The remaining ground rule will play a key role in the argument of section 4. I argue that, if we take all issues concerned with phenomenal consciousness off the table, we must allow for the possibility of finding a computational model of human thought that accounts for everything else. This proposal is justified by the following observations:

1. It’s the working assumption that much cognitive-science research is based on. The computational revolution in psychology fifty or sixty years ago made one thing clear: that for the time being the big unsolved problem of psychology would no longer be explaining how the unconscious could work, but explaining how anything *conscious* could emerge. After fifty years that’s more or less still where we are; there are those who think that there can be a computational account of phenomenal consciousness, and those (the majority) who

---

<sup>21</sup>I allude once again to “The Library of Babel.”

<sup>22</sup>Cf. (Culbertson 1956), although Culbertson was talking about a somewhat different set of robot-control mechanisms. He pointed out that they were “uneconomical,” which must be the greatest understatement of all time.

<sup>23</sup>In (Block 1978), Block points out that ... If it [the strategy tree] is to ‘keep up’ with current events, the job [of rebuilding it] would have to be done often” (p. 295). He hasn’t thought through just how difficult this would be.

think some extra ingredient will have to be added to the mix. Still, the research that has been done without anything like a consensus (some would say hint) about what that ingredient could be has been enormously successful.

2. Suppose it is impossible to model the human brain computationally, in the sense that no matter how fine-grained one's model of the brain's neurons, synapses, vesicles, glands, glial cells, etc., something crucial is always left out. (The brain might be a quantum supercomputer employing an unimaginable number of superposed wave-functions.) Then it's difficult to see how scientific psychology is even possible. But if there *is* an approximate computational model of the Aunt Bertha's brain, no matter how opaque, it could surely be adapted to simulate her taking a Turing test.
3. If there is no computational model of Aunt Bertha, or some other convenient personage, then the entire discussion is unmotivated. Turing proposed the Test as a thought experiment involving a *computer*. If the entire enterprise of modeling the human brain/mind computationally is doomed, then the experiment could never be performed anyway.
4. The HTP is itself a computational model, albeit not a practical one. (And not very interesting either, since it covers only one hour of Aunt Bertha's life, an hour in which she has agreed to do almost nothing but have an instant-message conversation with the interrogator.)

So ground rule 3 is that we leave open the possibility of there being a computational theory that is at least a partially correct account of psychological states, in that it may (many would say must) leave phenomenology unexplained. If this possibility is rejected, then it's hard to see what there is to discuss. If every computational model of mental states is doomed to fall short, then I'm willing to grant that a program that passed the Turing Test would be unintelligent, for roughly the reason that I'd be willing to grant that a perpetual-motion machine would get lousy gas mileage. On the other hand, if some computational theory *is* correct, then there must surely be a computer program based on that theory that can carry on an hour-long conversation as if it were "Aunt Bertha." It might require a large number of processors and a lot of memory, but even if the numbers turn out to be close to the numbers of neurons or synapses in the human brain, they will still be much smaller than the number of entries in the sensible-string table. (See section 3.1.)

The following terminology will prove useful in what follows. If  $M$  is an accurate computational model of a person  $x$  (a hypothetical rather than a real person, in all likelihood), then we'll let  $C_x$  be a function from states of  $x$  to states of  $M$ , such that  $C_x(s)$  is the state  $y$  of  $M$  that corresponds to state  $s$  of  $x$ . Whatever  $x$  believes or intends in state  $s$  is also believed or intended by  $M$  in state  $C_x(s)$ .<sup>24</sup> Whether  $M$  can experience what  $x$  would or remember experiencing

---

<sup>24</sup>There might be issues of wide vs. narrow content here (Botterill and Carruthers 1999), but they probably take a back seat to problems raised by the fact that  $x$  and her world are fictional.

something  $x$  might have experienced is moot, but it can certainly speak as though it *believes* it remembers experiencing things. Turing made that clear in (Turing 1950). What this all amounts to is an endorsement of (the possibility of the truth of) *functionalism* about beliefs and desires (Botterill and Carruthers 1999).

In this paper I often use verbs like “believes” or “expects” about computer programs of various sorts. Those who doubt that the programs in question (based on table lookup or some other suite of techniques) can actually exhibit such propositional attitudes should put scare quotes around them (and read “knows,” for instance, as “behaves for all the world as if it knows” when used of a suspect computational agent). Other readers will have no doubts that the program actually has beliefs. I don’t want to take a position in this dispute, at least not prematurely, but I also don’t want to clutter the paper up with the literary equivalent of air quotes. I could deploy some sort of special quotation device to try to satisfy everybody, but instead I will just use the unadorned verb in all but the most extreme cases. Don’t worry; I will avoid using this convention to beg the question at moments of crisis.

While we’re on the subject of psychological modeling, I should say a word about nondeterminism. Although Block seemed to include nondeterminism in his first description of the HTP, it is not necessary to the argument I will make in section 4. Nonetheless, it is impossible to treat a *confederate* as a deterministic system. Even if they are as deterministic as you like, they are receiving other inputs besides the sentences being typed at them as part of the Turing test, and these inputs will surely exert enough influence to alter their behavior slightly. I believe it’s in the spirit of Turing’s idea to limit the effect of such extraneous inputs by having confederates sit alone in a small, dull, quiet room, and having examinees believe they, too, are sitting in a similar room.<sup>25</sup>

In any case, we can’t require of a psychological model of (e.g.) Aunt Bertha that if the initial state of the woman is  $s_0$ , and the initial state of the model is  $y_0 = C_{AB}(s_0)$ , and the two receive the same typed inputs from the interrogator, that they will wind up in corresponding states. More formally, let  $TS_M$  be the state-transition function for the model  $M$ :  $TS_M(y, I)$  is the state  $M$  goes into if it receives input sequence  $I$  in state  $y$ . (Of course, after each input  $\in I$  there is an output from  $M$ , but we’ll ignore that.) The transition function  $TS_{AB}$  for Aunt Bertha can’t work the same way, because of the nondeterminism. Instead we’ll have to let  $TS_{AB}(s, I)$  be the *set* of states Aunt Bertha could wind up in if she starts in state  $s$  and receives inputs  $I$ . We could get quite technical here about the probability distribution on  $TS_{AB}(s, I)$ , but instead we’ll just stipulate that none of this is known to the judge(s). A program  $M$  passes the Turing Test if it is a faithful psychological model of a possible person; that is, as far any judge can tell, there *could be* a state  $s_0$  of some (non-demented)  $x$  such that state  $TS_M(y_0, I) = C_x(s)$  for some “reasonably probable” state  $s \in TS_x(s_0, I)$ .

---

<sup>25</sup>It’s odd that no one has, as far as I know, raised this issue before. If the surroundings of the participants are not made uniform the interrogator might be able to figure out who’s who by asking the participants to describe the location where they’re sitting.

We can get away with a deterministic model of a nondeterministic person only because the TT is not repeated using the same judge, and the judges are not allowed to confer before announcing their judgements. Otherwise, the fact that the model always makes the same responses to the same series of inputs would give it away. If a judge is allowed to request to talk to some interlocutors again, we can treat that as a continuation of the original interview, with or without adjustment of the time limits. If judges are allowed to confer, or if examinees never know whether they've talked to their current interrogator before, we could treat the entire series of interviews as one long interview, and treat the set of judges as a single "jury" (a word Turing used in (Braithwaite et al. 1952)). But I will assume all such adjustments are unnecessary.

Let me clear up one confusion about deterministic programs masquerading as nondeterministic. Suppose the judge asks an examinee to play a series of games of tic-tac-toe (after negotiating a coding convention for typing moves). The examinee can be deterministic and still make different moves for different games in the series, because it's never in the same state at the start of each game. (It remembers playing the earlier games.) One way to think of it is that, if dice must be thrown, they can be thrown when the program is constructed rather than when it is run. The idea that computers have to do the same thing whenever they are in superficially similar situations is akin to the fallacy that a computer would give itself away by never making arithmetic errors, being intolerant of contradictory information, and so forth.

There is no particular reason to *require* psychological models to be deterministic except to simplify the exposition. The argument here is meant only to establish that a model *could* be deterministic without an interrogator being able to detect that fact.

## 4 The Refutation

We are finally ready to work on two arguments that refute Block's claim. Both of them come down to this: The state space for any automaton becomes finite if its lifetime is made finite, and this state space can itself be considered an enormous automaton, which is just as intelligent as the one we started with.

### 4.1 Argument One: Why The Existence of HTPS's Proves Nothing.

In section 2, I introduced the terms "HTPL" and "HTPS" to describe two different but behaviorally equivalent forms of the Humongous-Table Program, one that looks strings up in a table, the other that searches a strategy tree. Most descriptions of the HTP are in terms of HTPL. Braddon-Mitchell and Jackson focus on the HTPS and describe it in a somewhat unusual way, which suggests some interesting insights. They suppose that the game tree, in the form of figure 1, is "inscribed" on a "chip" (Braddon-Mitchell and Jackson 2007, p. 115). One

pictures some sort of microfiche that a computer then examines visually. But that's not the way diagrams like these are actually inscribed on chips.

Figure 1 is isomorphic to a *finite-state machine* (FSM) (Kam 1997) of a simple kind in which states are never repeated.<sup>26</sup> But do not read “simple” as “trivial,” because any digital computer that is allowed to run for only a fixed amount of time will go through a finite number of states, and if that computer embodies a correct psychological theory of, say, Aunt Bertha, then it is almost certain that it will never repeat a state. Hence this machine (the possibility of whose existence, by ground rule 3, we must seriously entertain) would have possible behaviors over a bounded period isomorphic to an FSM with a tree-shaped state-transition graph.

One problem with the FSM of figure 1 is that it is defined over an “alphabet” of  $10^{367}$  “symbols,” the possible input strings. It's more in the spirit of how such things are actually built for the alphabet to be visible Ascii characters, plus space (written “\\_”) and the special mark ##. We achieve this by adding new states, as shown in figure 3. C-nodes are as before, and in particular we can still associate entire (timed) strings as the output at a C-node. But at a J-node the judge's input is handled one character at a time. Each character triggers a transition to a gray intermediate state, called an *A-node* where the machine outputs nothing (i.e., an empty string, which we could write  $\epsilon$  if we had to) and merely accepts the next character. Only when the human inputs ## (or types a character that is illegal given the rules on strings) does a transition to a C-node occur. With this convention, we have had to stretch the input "It's so sunny today" so only the "It's" and the final "y" are visible. Similarly for "Looks like rain", which has been “redacted” to "Lo...in".

Let me reemphasize that this transformation leaves the number of C-nodes (those in which the machine produces a nonempty string or "##") unchanged. Any restrictions we introduce on interrogator input to reduce the number of legal inputs affects a machine such as that of figure 1 by shrinking its alphabet; and affects a figure-3 machine by increasing the number of transitions from A-nodes to C-nodes. For instance, if we impose a rule that there can be no more than two occurrences of the letter “x,” and the interrogator types "Those Red Sox box seats in the sportsple", then if the next letter is x it's equivalent to ##. (If "Sox" is not in the common-word dictionary, and neither is any word beginning "sportsple", and the number of characters allowed for forming uncommon words were reduced from 40 to 12, then *any* character would trigger a transition to a C-node.)

FSMs are inscribed on chips in the following way: Some class of physical pattern is chosen to represent the states. In the real world, these patterns are usually bit strings, but some other way of encoding the information could be chosen, using the unusual sorts of infinitely divisible matter envisioned by Block, so long as the patterns fall into a finite number of classes that are strictly distinguished. There must in addition exist components that combine the next input with the pattern representing the current state in order to produce the next state and the

---

<sup>26</sup>When a leaf state is reached, the FSM halts.



output strings.<sup>27</sup> In the real world these components are made of silicon, but one could use other forms of matter. In current industrial practice the components are almost always driven by a central clock, but there are many techniques for eliminating the synchronizing clock, resulting in asynchronous circuits (Smith and Di 2009).

If an FSM has  $N$  states, then it takes at least  $\lceil \log_2 N \rceil$  bits to represent them, assuming they are all behaviorally distinct, which is obviously true in the case at hand. In practice, many more bits than this minimum are used, because bits are cheap, and it's worth using a few more than necessary to achieve conceptually simple circuits. So we might use a 7-bit register to represent an Ascii character, even though only 96 of the 128 possible bit patterns will ever occur; for a long string of characters the bit wastage adds up, but normally we don't bother compressing it out (or the redundancy from other sources).

Of course, we have no idea what techniques would be used in Block's infinitely divisible region of spacetime, but it would be stretching the notion of conceptual possibility to the breaking point to imagine that information-theoretic bounds would no longer apply. So the state of the FSM would have to correspond to some physical state set with enough variability to represent  $\lceil \log_2 N \rceil$  bits of information; and the transitions would have to be mediated through some physical device for selecting the next state in the state set. In what follows I will refer to this as a "bit string" mainly out of desperation, lacking any other concrete image of what it could be (besides switching to a base other than 2).

Although  $10^{17,500}$  (the total number of C-nodes in the HTPS, whether implemented using figure-1 conventions or figure-3 conventions) is a really scary number, the number of bits we need (assuming base 2) is its *logarithm*, which is only about 50,000 — about 6 kilobytes of storage! However, the money we would save on representing the state we would pay back for the circuitry required to transition from one bit string to the next, because the expected number of gates required to compute a random Boolean function of  $n$  bits is an exponential function of  $n$  (Wegener 1991).

Let's look at ways of being more parsimonious. Suppose the examinee is implemented using a Turing machine. Real computers use external storage devices, or, nowadays, "the cloud" (Furht and Escalante 2010) to take on board an indefinite amount of storage. A Turing machine uses an infinite tape. Whatever computation can be done by a random-access computer in polynomial time can be done by a Turing machine in polynomial time, so if we make the tape head zippy enough the Turing machine can spit out a (timed) string in the time allotted for the TT.<sup>28</sup> Normally the tape heads have no velocity as such; they are mathematical abstractions, and the time they take is measured in state transitions. But if we assign a velocity, the portion of the

---

<sup>27</sup>This is related to the function  $TS$  described in section 4.1, but that one ignored  $O$ , and took a *series* of inputs as argument.

<sup>28</sup>It is, of course, just a coincidence that Turing's name is on both the Turing Test and the Turing machine; he never linked the two, if you don't count vague allusions.

tape that can be reached in an hour-long conversation becomes finite. I'll use the term *tape allotment* for such a finite tape segment.

We'll actually assume the machine has 4 tapes: for input, output, work space, and randomness.<sup>29</sup> The judge's input comes in via the input tape. The output is written on the output tape. (We again duck the issue of the exact format of timed strings.) The work tape is used for all storage. Although it is not necessary, it will be convenient in what follows to assume that the workspace tape is not necessarily blank when the machine starts. If the machine is controlling an Aunt Bertha robot asked mid-career to participate in the TT, then the workspace tape might contain the information accumulated by "Aunt Bertha" before she walks into the examining room.

That leaves the random-input tape. It is a common device in the theory of computing to make a Turing machine random by assuming a read-only tape containing a purely random infinite series of bits. As discussed in section [[ ]], we can then make such a machine deterministic by using the same tape over and over. That is, pick a random infinite string of bits, and chop off the finite portion that can be reached in an hour given a convenient tape-head velocity. Assume this exact tape allotment is consulted every time the examinee is examined by an interrogator.

We now have a "finite Turing machine" (FTM). Its state space is the state space of the original machine  $\times$  the possible configurations of the four tapes, including the tape-head position. It can be considered a finite-state machine whose state may be represented by a bit string consisting of the binary representation of the current state number followed (for each tape) by the contents of the tape allotment and the binary representation of the position of the tape head (except that the random-tape contents, which never change, can be omitted).

One can picture a great many Turing machines that can pass the Turing Test (by ground rule 3), and perhaps there's one that organizes its bits into structured representations of logical formulas. That is, its workspace tape might initially contain the formulas `married (me, husb)` and `name(husb, "Herb")`. There are intermediate states with bit patterns stating

```
episode(e303, conversation(me, friend1023))
topic(e303, weather)
name(friend1023, pradeep)
```

Or, if you prefer, the bit patterns could store weights on artificial neural networks, weights that vary over the course of the conversation.

Nothing in a picture such as figure 3 rules out the possibility that the computations performed on these bit patterns are *exactly the same* as those in an open-ended model of Aunt Bertha,

---

<sup>29</sup>Using multiple tapes is a convenient device that doesn't change the computational power of Turing machines (Homer and Selman 2011, ch. 2).

except for being finite, but this is a constraint imposed by the bound on the length of the conversation.

Hence huge HTTPS's such as those schematized in figures 1 and figure 3 are not necessarily trivial in any sense. That's because the way the state transitions from a node to one of its children work is underdetermined.

## 4.2 Argument Two: Why The Existence of HTPL's Proves Nothing.

However, this argument leaves the possibility that a *particular* instantiation of HTTPS is trivial, namely, an HTPL. Braddon-Mitchell and Jackson force us to focus on HTPL's when they phrase their critique as the claim that the HTP "... is *so* like all the cases where we feel that someone lacks understanding: ... someone who cannot give you the square root of a number other than by looking up a table of square roots is someone who does not fully understand what a square root is" (p. 117, emphasis in original).

*This* argument does not work; it is a simple instance of a fallacy. The HTPL is not like a person who looks something up any more than a computer is like a person who follows instructions; these are just heuristic images. The fallacy in question, which is tiresomely common, is *misplaced anthropomorphism*. It consists of choosing some part of a computer to identify with a person, and then imputing to the computer the mental states (or lack thereof) a *person* would have if they did the job the computer does.<sup>30</sup> It seems to me that simply identifying the unsupported anthropomorphism is enough to defeat it, but if that doesn't convince you there is also the problem of choosing *which* part of the computer to pretend is a person; computers consist of layers of *virtual machines*, so it may be impossible to pick the one to anthropomorphize (Sloman and Chrisley 2003). The Java virtual machine (JVM) (Lindholm et al. 2012) consults a program that may be viewed as a table of byte codes in order to decide what to do next, but below that is another machine that looks up instructions in a table in order to execute the JVM.<sup>31</sup>

The HTPL itself is a virtual machine, the interpreter for a special-purpose programming language, whose programs are the possible key-response tables. Now the homunculus who is looking things up in a table drops out, and we are back to thinking of the program as a finite-state machine, whose states are represented as strings of interrogator inputs separated by occurrences of `##`.<sup>32</sup>

---

<sup>30</sup>Another example is Searle's (1980) "Chinese Room" argument. One reason it is so easy to fall into this trap is that the inventors of the first computers resorted so often to words such as "memory" to describe pieces of these new things, and we've been stuck with them ever since. But I confess that in teaching intro programming I get students into the right mindset by pretending the computer is a "literal-minded assistant" or some such thing, that variables are "boxes" this assistant "puts numbers in," and so on.

<sup>31</sup>This may or may not be the "real" machine, depending on whether machine language is executed by a microcode interpreter. And if the computer has several "cores," should we think of it as a committee?

<sup>32</sup>Recall that in section 2 we "optimized" keys by removing the examinee's contributions to the dialogue.

We now return to refuting Block’s argument. Suppose that  $M$  is a psychologically accurate computational model of Aunt Bertha. Using the notation developed in section 3, there is a function  $C_{AB}$  from psychological states of Aunt Bertha to states of  $M$ , such that every time  $M$  is in a state  $C_{AB}(s)$ , its behavior for the remainder of the test is a plausible behavior for Aunt Bertha if she starts in state  $s$ . Our ground rule 3 allows us to posit that such an  $M$  might exist, and would be as intelligent as Aunt Bertha, and would have mental states if she does. The only thing it might lack is phenomenal consciousness, although either way she would behave as if she had it.<sup>33</sup> I’ll use the term  $M_{AB}$  to designate this Aunt Bertha model. Let’s think for a while about the consequences of our assumptions about  $M_{AB}$ .

It may seem that there is a huge difference between  $M_{AB}$  and the HTPL, but it’s only the sort of difference created by optimizing compilers all the time. That is certainly not obvious, but consider the transformation called *loop unrolling*, which involves reducing the number of iterations of a loop (sometimes to zero) by replacing one or more iterations with simple if statements. To take a simple example,<sup>34</sup> expressed as a recursion for reasons that will become clear:

```
define simple_loop(k)
  {
    if k>0
      then {
        A;
        simple_loop(k-1)
      }
  }
```

then the loop can be rewritten

```
define simple_loop(k)
  {
    if k>2
      then {
        A;
        A;
        A;
        simple_loop(k-3)
      }
    else if k>0
      then {
        A;
        simple_loop(k-1)
      }
  }
```

---

<sup>33</sup>Of course, some people feel that it is absurd to deny a creature phenomenal consciousness if *it* doesn’t seem believe it lacks anything (Dennett 1978b).

<sup>34</sup>For the syntax of the programming language used in what follows, see appendix A.

```

    }
}

```

Why would one bother making the program longer this way? Normally one wouldn't. But if it's the inner loop of some supercomputer program for which performance is critical, speedups can be had by avoiding tests and decision points, thus filling the pipelines of high-performance CPU chips. (It will become clear shortly why I choose to express loops in this style, as programs that call themselves recursively as their last action.)

We can show formally that there is a sequence of meaning-preserving transformations from  $M_{AB}$  to a version equivalent to an HTPL. It's worth going through this carefully, because it is a central part of my argument. Any realistic  $M_{AB}$  will be a highly parallel program, given the way computers are evolving. But in what follows I will assume that we can eliminate all such parallelism by speeding up a single processor.<sup>35</sup> I will also assume that no "interrupt" plays an essential role in the operation of the program, except of course the interrupt that ends the program after an hour. For instance, "Aunt Bertha's" attention is not suddenly diverted by a bluebird flying into the window of the examining room (a simulated room, unless  $M_{AB}$  is controlling a real robot). Instead, she remains steadily focused on the screen containing the dialogue with the interrogator. So we can idealize the program as consisting of one outer loop, during every iteration of which "AB" reads a query from the interrogator and reacts to it. Formally, we will write the program as

```

{
  Settle down and initialize;
  aunt_bertha_loop(KB0)
}

```

where the loop is defined thus:

```

define aunt_bertha_loop(KB)
{
  let R = read (next query from judge) in
  {
    KBnew ← react to R using KB;
    aunt_bertha_loop(KBnew)
  }
}

```

Here KB is the program's knowledge base.<sup>36</sup> I do not mean to endorse an extreme version of the representational theory of mind here (Fodor 1975). Not all of "Aunt Bertha's" beliefs have to

---

<sup>35</sup>A set of deterministic processors acting asynchronously in parallel would be nondeterministic, and this nondeterminism would be eliminated when we switch to a single processor. But I argued above (section 3.3) that a judge would be unable to tell the difference between a deterministic and nondeterministic program.

<sup>36</sup>It may seem unusual to compute a new knowledge base rather than make changes to the old one, but it's

be encoded explicitly and linguistically in the KB. For instance, she might be a misanthrope, but her attitude toward humanity might be implicit in the largish chunk of code we have hidden under the rubric “*react to R.*”

Of course, a complete model would have to deal with Aunt Bertha’s long-term goals, desires, hopes, and dreams, but we can idealize all that away, because we are interested in the behavior of the model only in the following context (called “context T” in the statement of theorem 1, below):

```
{
  Settle down and initialize;
  aunt_bertha_loop(KB0)
}
in-parallel-with
{
  wait(1 hour);
  exit
}
```

The construct  $c_1$  `in-parallel-with`  $c_2$  means to execute  $c_1$  and  $c_2$  simultaneously, until both are done, or until one of them executes the special command `exit`, which causes the program to terminate.

I will relegate most of the further technicalities to the appendix, and just summarize my conclusion as a theorem:

**Theorem 1** *In context T, the call to `aunt_bertha_loop` may be transformed via a series of meaning-preserving transformations into an equivalent sequence of `if` statements that is isomorphic to the HTPL.*

See the section B for the proof.

The question is then: At what point in this series of transformations does the intelligent program cease to be intelligent? Remember that its abilities do not change in any way. If Aunt Bertha was a Phi Beta Kappa who went on to get a Ph.D. in mathematics, then  $M_{AB}$  is a pretty sharp cookie, and so, it seems to me, are the transformed versions of  $M_{AB}$ , right up to the HTPL version. If she’s just a regular person who likes to talk about Tolkien, then so does  $M_{AB}$  and

---

a standard move made for technical reasons; the compiler is supposed to eliminate any inefficiencies that result from this device. I will take this opportunity to insert the standard disclaimer about the term “knowledge base”: It should really be called the “belief base,” but for some reason that term hasn’t caught on.

the HTPL version of  $M_{AB}$ . There is *no* plausible point at which the computational models lose whatever mental properties they are posited to have at the outset, except insofar as *any* condemned person may be said to lose their faculties when their time is up.

One might object that the HTPL never repeats a state as it works its way down the strategy tree. But, as I mentioned in section 4.1, this is a feature of just about any complex computation. Or that the HTPL will have exactly the same conversation repeatedly if you say the same things to it, with no memory of the previous iterations. But *any* program — in fact, any physical system — will have this feature, if after an hour you can reset it to its original state. The only real difference is that the HTP *must* be restarted, because unbeknownst to it its lifetime is very short.

So when Braddon-Mitchell and Jackson say that, “The state that governs the responses to inputs early on plays the wrong kind of role in causing the state that governs the responses later on...,” it’s clear that they’ve drawn too strong a conclusion: the role differs in detail, but not “in kind” from the role of state transitions in a longer-lived program such as  $M_{AB}$ . If it weren’t for the bizarre economic constraints in the fanciful universe of the HTA, which are obviously quite different from the constraints on real computers that make optimizing compilers worthwhile, the transformations required to get from  $M$  to its finite-state counterpart would not make any sense. But it’s the proponents of the HTA that are responsible for creating this universe; the rest of us just have to imagine living in it.

We have not only arrived at a direct equivalence between the HTPL and  $M_{AB}$ , but we have provided a way for the HTP to be created that does not require the action of a swarm of intelligent beings, which (Ground Rule 2) we forbade. The transformations that take us from  $M_{AB}$  to the HTPL are perfectly mindless.  $M_{AB}$  has mental states, but the transformations that take it to HTPL do not. Of course, we still have the problem that carrying the transformations out would take eons; but if you accept the conceivability of the HTPL you must accept the conceivability of the transformations that would produce it.

It might be claimed that, although there is no sharp point in the transformation of the  $M_{AB}$  into a tree of **if** statements where it ceases to be intelligent, the series of changes have the effect of replacing real “cogitations” with mere dispositions to behave in various ways. Eventually, there is nothing left but such dispositions. Hence the program has mental states only if analytical behaviorism (Braddon-Mitchell 2009) is correct and all talk of mental states can be replaced by talk of dispositions to behave. But behaviorism was refuted fifty years ago, was it not?

The principal critique of behaviorism to which this objection alludes is that it proved to be impossible to actually analyze any realistic mental-state term into statements about behavior or behavioral dispositions; attempted analyses had too many disjunctions, which seemed to refer ineluctably to other mental-state terms (Chisholm 1957; Geach 1957; Braddon-Mitchell 2009). The problem is that this critique fails under the strange conditions the tree of **ifs**

operates under. Only a finite number of things can happen to  $M_{AB}$  because, even though most of the time  $10^{17,500}$  would be treated as practically infinite, we have decided to pretend it's just another number. Furthermore, whatever desires  $M_{AB}$  may have beyond the TT, she has shelved them for an hour, which, unbeknownst to her, is her last.<sup>37</sup> Hence we *can* produce a finite analysis of every mental-state term that applies to “Aunt Bertha” into a vast but finite disjunction of statements about how she would behave.

One might object that my argument works only if we assume that the HTPL correspond to a psychological model  $M$  of a plausible real person. Ground rule 3 allows us to suppose that there might be a great number of computational models indistinguishable from people in the context of the TT; my argument in this section is that the corresponding HTP may be viewed as an optimized version of such a model. But in section 3.2 I pointed out that the space of randomly assembled FSMs is of Borgesian proportions. Almost none of them correspond to any obvious  $M$ . Most of them babble meaninglessly, but there are few “autistic savants” capable of amazing mental feats. Would they have psychological states?

For most of them, the answer is no. But the topic under discussion is programs that can pass the Turing Test. These must be very good simulacra indeed of plausible human beings. If we found a robot that behaved like the extreme autistic savant, it might put a lot of mathematicians out of work, but it couldn't pass the Turing Test. If we restrict our attention to instances of HTP that *can* pass it, how can we possibly dismiss the possibility that a candidate instance is a hyperoptimized version of a respectable mental model?

Given these ideas on what it means for an HTP to be in a certain psychological state, we can be precise about the sense in which the anthropomorphism discussed at the end of section 4.1 is wrong-headed. Contrast these two statements:

1. The program remembers being asked to solve three arithmetic problems.
2. The program remembers looking up its responses in the sensible-string table.

We can give a precise meaning to these statements, analogous to the way we gave a meaning to remembering having been told the first name of the interrogator. But the HTP is never in the second state. That is, there is no state of  $M_{AB}$  that corresponds to memory 2. This is hardly surprising; *you* have no memories of transmitting signals from your hippocampus to your frontal lobe.

---

<sup>37</sup>Of course, she can *discuss* them, and probably will if the judge brings them up.

## 5 Conclusion

Our conclusion is that the Humongous-Table Argument fails to establish that the Turing Test could be passed by a program with no mental states. It merely establishes that if real estate is cheap enough, the lifespan of a psychological model is predictable, and all its possible inputs are known, then the model may be optimized into a bizarre but still recognizable form: the Humongous-Table Program. If the model has mental states then so does the HTP. Furthermore, this is the only legitimate and plausible way ever suggested to produce the HTP without violating ground rule 2, that sentient beings not be involved in the construction.

Of course, if you reject my ground rule 3, the premise that cognitive science is onto something, and that computational models could turn out to have some validity, then my argument will not convince you. You don't accept the antecedent of the conditional, so you needn't worry about the consequent. But in this case the Humongous-Table Argument is redundant. You don't think the enterprise Turing was interested in, of creating intelligent programs that could pass his Test, is feasible.

On the other side, perhaps my attempt to imagine the Humongous-Table Program in some detail (see ground rules 1 and 2) will convince you that one's initial tendency to accept the conceivability of a finite table of responses for controlling a TT examinee is a mistake. Perhaps the thing is simply too huge to imagine coming into existence; or you're tripped up by the fact that no one could ever *verify* that an examinee was actually an instance of the Humongous-Table Program, and that it wasn't simply a transmission medium for the mental states of electron-sized aliens. If you're in this category, the HTA cannot really get started, and so is irrelevant.

It may seem that I have used a sledge hammer to smash a flea, that such an odd argument as the HTA doesn't deserve treatment at this length. But the argument occupies a strategic position whose importance is out of proportion to its plausibility. It is the *only* argument for the insufficiency of the Turing Test as evidence for the intelligence of an agent, except for arguments that purport to cast doubt on the intelligence of *any* program connected the way the computer is while taking a Turing test. I am thinking here of Searle's "Chinese Room" argument (Searle 1980) or Harnad's argument from symbol grounding (Harnad 1990). If such an argument goes through, then *every* program that passes the TT fails to be truly intelligent, so of course the Humongous-Table Program does too. But the HTA is meant to show *directly* that intelligence does not follow from success in the TT while leaving open the possibility that some program could pass it "the right way," by having the right kind of data structures, algorithms, neural networks, and other psychologicistic features. Its importance is magnified by the fact that just about everybody accepts it. Most people share the intuition that a table-lookup system (or something close to it) is feasible, and just couldn't be intelligent. I hope I have at least gotten some doubt to creep into your mind regarding this intuition.

I hope also that my remarks are not taken as a defense of the Turing Test. I agree with the critique of (Hayes and Ford 1995; Lenat 2009) that the influence of the Test on AI research has been mostly negative. Passing the Test is obviously not necessary for a program to be intelligent, and although it is hard to doubt that passing it is sufficient to verify intelligence, it looks as if doing so would require a program to be a close model of human psychology (French 1990). This is demanding more than Turing had in mind.

## A A Simple Programming Language

The algorithms used in discussing transformations leading to the HTPL are expressed in a simple programming language. There is no distinction between commands and expressions; commands are just expressions with side effects. Assignments are written *var*← *val*. Sequences of statements are written as

$$\{ s_1; s_2; \dots s_l \}$$

Each  $s_i$  is an expression. Braces are used to indicate eliminate ambiguity in grouping.

Conditionals are of the form *if*  $e_1$  *then*  $e_2$  [*else*  $e_3$ ]. (The *else* is optional.)

Functions are defined thus:

```
define name(—parameters—)  
  e
```

where  $e$  is an expression, often a sequence.

Function parameters become locally bound variables when the function is applied to arguments. The other way to bind variables is

```
let  $v = e_1$  in  
   $e_2$ 
```

which evaluates  $e_1$ , then binds  $v$  to its value during the evaluation of  $e_2$ .

See section 4.2 for discussion of the constructs defined using *in-parallel-with* and *exit*.

Pseudo-code is written with italics.

## B Proof of Theorem

I will use the term *partial evaluation* for the behavior-preserving transformations used to prove theorem 1. The term is used to refer to performing some of a program’s operations in advance when some of its arguments are known (Jones et al. 1993).

One of the transformations covered by the term “partial evaluation” is *constant folding* (Allen and Kennedy 2001), the process of substituting a known (usually numerical) value of a variable for all its occurrences, and then simplifying. Another is *function unfolding*, in which the definition of a function is substituted for one of its occurrences (and, again, propagating the resulting simplifications).

The simplifications to be propagated are straightforward, except for random-number generation. At every point in the program where a random number is needed, we perform *randomness elimination*. This is one of two transformations, depending on how the computer is designed:

1. If the program actually relies on pseudo-random numbers (Knuth 1998, ch. 3), the random-number generator is run at compile time (which changes the stored “seed” needed to produce the next pseudo-random number).
2. If the computer has an actual “generate random number” instruction, then we generate one. By definition, the number depends on nothing in the program, so running it in advance is equivalent to running it in real time.<sup>38</sup> (I doubt there are any computers in existence today that actually have such an instruction, but the Manchester Alpha machine, for which Turing co-authored a manual (Turing and Brooker 1952), had an instruction of this kind.)

Partial evaluation will also include three new transformations. The first is called *input anticipation*. It consists of transforming any piece of code with the form

```
let  $r = (\text{read some source } \dots)$  in  
   $A[r]$ 
```

in a situation where the possible values read are a finite set  $\{v_1, \dots, v_n\}$  into

---

<sup>38</sup>If we supply a special input channel from which random numbers are read, analogous to a tape containing random bits for a Turing machine (section 4.1), then we can just treat randomness elimination as a special case of input anticipation.

```

let r = (read some source ...) in
  if r = v1
    then A[v1]
  else if r = v2
    then A[v2]
  ...
  else if r = vn
    then A[vn]

```

where  $A[v]$  is  $A$  with  $v$  substituted for all free occurrences of  $r$ .

The second new transformation is *let-elimination*. Any expression of the form

```

let v=e1 in
  e2[v]

```

(which binds variable  $v$  to value  $e_1$  while evaluating expression  $e_2$ , which contains zero or more occurrences of  $v$ ) may be transformed into

```

v1 ← e1
e2[v1]

```

where  $v_1$  is a variable that occurs nowhere else in the program and  $e_2[v_1]$  is  $e_2$  with every free occurrence of  $v$  replaced by  $v_1$ .

The third new transformation is *if-raising*. Code of the form

```

if P1 then
  { c1;
    if P11 then { d11 }
    else if P12 then { d12 }
    ... else if P1,k1 then { d1,k1 }
  }
else if P2 then
  { c2;
    if P21 then { d21 }
    ...
  }

```

```

...
else if  $P_k$  then
  {  $c_k$ ;
    if  $P_{k1}$  then {  $d_{k1}$  }
    ...
    else if  $P_{k,k_k}$  then {  $d_{k,k_k}$  }
  }

```

may be transformed into

```

if  $P_1$  then  $c_1$ 
else if  $P_2$  then  $c_2$ 
...
else if  $P_k$  then  $c_k$ ;
if  $P_1$  and  $P_{11}$  then {  $d_{11}$  }
else if  $P_1$  and  $P_{12}$  then {  $d_{12}$  }
... else if  $P_1$  and  $P_{1,k_1}$  then {  $d_{1,k_1}$  }
else if  $P_2$  and  $P_{21}$  then {  $d_{21}$  }
...
else if  $P_k$  and  $P_{k1}$  then {  $d_{k1}$  }
...
else if  $P_k$  and  $P_{k,k_k}$  then {  $d_{k,k_k}$  }

```

The idea is that if every clause of an `if-then-else` statement ends in an `if-then-else`, then those terminal `if-then-else`'s can be raised to the level of the original `if-then-else`, provided we add an extra condition to every `if-test`. For example, the last line mimics the last `else-if` clause of the original schema by adding the gating condition  $P_k$  that used to be there implicitly because of the nested control.

Be sure to note the (easy-to-miss) semicolon between the first  $k$  `if` clauses and the rest of the program. It means that after those first tests are run, control passes to the remaining ones without returning to the first group.

*Proof of theorem 1:* The call to `aunt.bertha_loop(KB0)` in context `T` may be transformed via partial evaluation into a list of `if` statements that is isomorphic to the HTPL.

*Proof:* The first step in transforming the program is to add an argument that records an upper bound on the amount of time the program has left.

```
define aunt_bertha_loop_t(max_time_left, KB)
```

```

{
  if max_time_left > 0
  then {
    let R = (read next query from judge) in
    {
      ⟨KB_new, T_C⟩ ← react to R using KB;
      aunt_bertha_loop_t(max_time_left - T_MJ(R) - T_C, KB_new)
    }
  }
  else exit
}

```

We call this version `aunt_bertha_loop_t`. Now the *react* statement returns the time  $T_C$  it took to type the output; and  $T_{MJ}(R)$  is the *minimum* time it would take the judge to type the string  $R$ . Adding the `if` statement doesn't affect correctness, because, assuming the initial value of `max_time_left` is  $\geq$  the actual amount of time remaining on the clock, then it obviously remains an upper bound in the recursive call to `aunt_bertha_loop`.

We ensure that this is indeed the case by replacing the original call to `aunt_bertha_loop` with

```

aunt_bertha_loop_t(1 hour, KB_0)
which is equivalent to
{
  let R = (read next query from judge) in
  {
    ⟨KB_new, T_C⟩ ← react to R using KB_0;
    aunt_bertha_loop_t(1 hour - T_MJ(R) - T_C, KB_new)
  }
}

```

(We do one function unfold, then simplify; because `1 hour > 0` evaluates to `true`, we can replace the `if` statement with its `then` clause.)

The statement binding  $R$  is the bit that reads what the judge types. There are  $10^{367}$  possible values for  $R$ . Because we are back in Unbollywood, where space and time cost essentially nothing, we can use input anticipation to introduce a  $10^{367}$ -way branch<sup>39</sup> after the input statement. The program has become the version shown in figure 4. Obviously, this is a crazy “optimization” to perform, but we do it only because of the strange economic pressures of Unbollywood.

---

<sup>39</sup>In this appendix I use the word “branch” to mean something different from the meaning explained in section 2: a decision point in a program, an “`if`” statement, conditional jump, or the like.

```

{
  let R = (read next query from judge) in
    if R = "###"
      then{
        print "###"
        exit
      }
    else if R = "It's so sunny today"
      then{
         $\langle \text{KB}_{\text{new}}, T_C \rangle \leftarrow \text{react to "It's so sunny today" using KB}_0;$ 
        aunt.bertha_loop_t(1 hour -  $T_{MJ}(\text{"It's...ay"}) - T_C$ , KB_new)
      }
    else if R = "Looks like rain"
      then{
         $\langle \text{KB}_{\text{new}}, T_C \rangle \leftarrow \text{react to "Looks like rain" using KB}_0;$ 
        aunt.bertha_loop_t(1 hour -  $T_{MJ}(\text{"Loo...ain"}) - T_C$ , KB_new)
      }
    else...
      about  $10^{367}$  more branches
      ...
}

```

Figure 4: Program after branch expansion

Now, within each branch, the values of both  $R$  and  $\text{KB}_0$  are *known*. Several consequences follow from this fact. The intricate structure of code I've summarized as “*react to ...*” can be simplified. Everywhere a test is performed that depends on the value of  $R$ , we can eliminate the dependency, discarding all paths through the code that are incompatible with what we know the value of  $R$  to be. When a random number is needed, we apply randomness elimination, changing the call to the random-number generator into a constant. (The output from a random-number generator is a number chosen from a uniform distribution. Often the outputs are squeezed into a different distribution using parameters available at run time; the values of these parameters are available during partial evaluation as well.)

There are only three results of this process we care about:

1. The variable  $T_{MJ}(R)$  in each branch of the *if-then-else* has become constant, so we can compute immediately the minimum time it would have taken to read  $R$ .
2. The characters that are typed by the “*react to*” code, and their times, can be collected

as a (timed) string  $S$ . The net behavior can be written as `print S`.

3. The variables `KB_new` and `TC` can be computed.

Hence, in each branch, we can replace the code written “*react to ...*” with `print S`, and the call to `aunt_bertha_loop_t` with the definition of `aunt_bertha_loop_t`, with constants substituted for its arguments. This expanded definition begins

```
if T > 0 then C ...
```

where  $T$  is a constant, the value of `max_time_left` in the call being expanded. In this first round of expansions,  $T$  is likely to be greater than 0 in virtually every call to `aunt_bertha_loop_t`, because a single exchange between the judge and the simulation is unlikely to take more than an hour.<sup>40</sup> So we can replace the `if` with `C`, which looks like

```
let R = (read next query from judge ) in
{
  <KB_new, TC> ← react to R using KB;
  aunt_bertha_loop_t(T - TMJ(R) - TC, KB_new)
}
```

$T$  and  $KB$  are *constants*, different in each branch. The result looks like figure 5, where these constants have been given subscripts.

When we’re done with all that, we start our series of transformations all over again on each new instantiation of `aunt_bertha_loop_t`, unfolding it, adding an `if` statement to branch on each possible value of the input, and partially evaluating each branch.

What we would like to do is apply `if-raising` repeatedly. But the `let`’s are in our way. This is where `let-elimination` comes in (not too surprising). In each branch we create a new variable,  $R_i$  for the  $i$ ’th branch; so that branch  $10^{367}$  will have the variable  $R_{10^{367}}$ . The result is as shown in figure 6.<sup>41</sup>

Each `read` can be subjected to input anticipation, and further expansion ensues. After the next round each clause of the outer `if` will be of this form:

---

<sup>40</sup>Although it’s hard to be completely sure of what happens in  $10^{367}$  branches.

<sup>41</sup>How come I haven’t had to treat `KB_new` and `TC` the same way I handled `R`? I should have, but it’s not necessary, because the name reuse doesn’t actually cause any confusion.

```

{
  let R = (read next query from judge) in
    if R = "##"
      then {
        print "##";
        exit
      }
    else if R = "It's so sunny today"
      then {
        print "Gee, the forecast was for rain";
        {
          let R = (read next query from judge) in
            {
              <KB_new, TC> ← react to R using KB1;
              aunt_bertha_loop_t(T1 - TMJ(R) - TC, KB_new)
            }
        }
      }
    else if R = "Looks like rain"
      then {
        print "My garden can use it.";
        {
          let R = (read next query from judge) in
            {
              <KB_new, TC> ← react to R using KB2;
              aunt_bertha_loop_t(T2 - TMJ(R) - TC, KB_new)
            }
        }
      }
    else ...
      about 10367 more branches
    ...
}

```

Figure 5: Program after further branch expansion

```

else if R = stringi
{
  print responsei;
  Ri ← (read next query from judge);

```

```

{
  let R = (read next query from judge) in
    if R = "##"
      then {
        print "##";
        exit
      }
    else if R = "It's so sunny today"
      then {
        print "Gee, the forecast was for rain";
        R1 ← (read next query from judge);
        ⟨KBnew, TC⟩ ← react to R1 using KB1;
        aunt.bertha.loop.t(T1 - TMJ(R1) - TC, KBnew)
      }
    else if R = "Looks like rain"
      then {
        print "My garden can use it.";
        R2 ← (read next query from judge);
        ⟨KBnew, TC⟩ ← react to R2 using KB2;
        aunt.bertha.loop.t(T2 - TMJ(R2) - TC, KBnew)
      }
    else ...
      about 10367 more branches
    ...
}

```

Figure 6: Program after applying let-elimination in every branch

```

if Ri = ## then
  ...
else if Ri = "It's so sunny today" then
  ...
else ...
10367 more branches
}

```

That means we can use if-raising, transforming the program into the form schematized in figure 7. In this figure, the “first if” has 10<sup>367</sup> branches; the second, (10<sup>367</sup>)<sup>2</sup>.

The program will gradually evolve into a gigantic list of if statements, which occasionally emits some characters to be sent to the interrogator, and along the way builds and preserves data

```

let R = (read next query from judge) in
{
  // First if
  if R = "##" then
    {
      print "...";
      R1 ← read (...);
      if ... Another huge bunch of choices
    }
  else if R = "It's so sunny today" then
    {
      print "...";
      R2 ← read(...);
      if ...
      ...
    }
  else ... ;
  // Second if
  if R = "##" and R1 = "##" then
    ...
  else if R = "##" and R1 = "It's so sunny today" then
    ...
  else if R = "##" and R1 = "Looks like rain" then
    ...
  ...
  else if R = "It's so sunny today" and R2 = "It's so sunny today" then
    ...
  ...
}

```

Figure 7: Sketch of program after applying if-raising to the outer loop.

structures (the KBs) for future use. Although rather bulky, the list is finite, because, even though `aunt_bertha_loop.t` is written as a potentially infinite recursion (which will be terminated by the alarm clock if necessary), the argument `max_time_left` is smaller for each recursive call. In every branch it eventually becomes the case that the if statement `if max_time_left > 0 then ...else exit` expands into `exit`.

Now, this list of if's is isomorphic to the HTPL. Each test is of the form

$$\text{if } R = \dots \text{and } R_{j_2} = \dots \text{and } \dots R_{j_k} = \dots \text{then}$$

(Treat  $R$  as if it were  $R_0$  and let  $j_0 = 0$ .) In this list of tests,  $k$  starts at 1 and there are  $10^{367}$  branches of that length; then it becomes 2 and there are  $(10^{367})^2$  branches of length 2; and so forth up to  $k =$  the maximal number of exchanges between interrogator and examinee that can occur in one hour.

This might as well be a (rather laborious) table lookup for the string corresponding to the conversation so far. At first we check for strings of length 1, then strings of length 2, and so forth.<sup>42</sup> These strings correspond exactly to the key strings used in HTPL. (See section 2 for why the length of a key string = the number of interrogator inputs so far.) QED

Please note the fate of the knowledge base as these transformations are made. Each version of `KB_new` reflects the acquisition of a few new beliefs as a result of one interchange with the judge, and of course the retention of old ones. Initially the facts `married(me, husband)` and `name(husband, "Herb")` might be stored in the cloud, so that if anyone asks for AB's husband's name, AB can respond "Herb". A new fact like `name(judge, "Pradeep")` might be added later. At some point the response "Herb" to the query "What's your husband's name?" or variants thereof occurs in a huge number of branches of the tree, and similarly for the query "What's my name, do you remember?". But as branches are closed off because they exhaust all the available time, these versions of the KB are discarded. If the transformation process runs to completion, eventually every possible way that any piece of information recorded in the KB might be reflected in AB's behavior *is* so reflected, and there is no longer any need for the knowledge base. We are in behaviorist heaven, where it really is the case that any fact about what the program believes can be expressed as an (incredibly large but finite) disjunction of dispositions to behave in certain ways.

## References

- Randy Allen and Ken Kennedy 2001 *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. San Francisco: Morgan Kaufmann
- Kenneth Binmore 2007 *Playing for Real: A Text on Game Theory*. Oxford University Press
- Ned Block 1978 Troubles with functionalism. In Savage (1978), pp. 261–325
- Ned Block (ed) 1980 *Readings in the Philosophy of Psychology*. Cambridge, Mass.: Harvard University Press. 2 vols
- Ned Block 1981 Psychologism and behaviorism. *The Philosophical Review* **90**(1), pp. 5–43

---

<sup>42</sup>If you really, *really* want the program to be isomorphic to the HTPL, you could transform it once again by converting it to a loop with an iteration-counting variable, adding a test for the appropriate value of this variable to every test of the `if` and replacing the semicolons with `else`'s. A transformation to accomplish this ("loop imposition"?) is left as an exercise for the reader.

- George Botterill and Peter Carruthers 1999 *The Philosophy of Psychology*. Cambridge University Press\*
- David Braddon-Mitchell 2009 Behaviourism. In Symons and Calvo (2009), pp. 90–98
- David Braddon-Mitchell and Frank Jackson 2007 *Philosophy of Mind and Cognition*. 2nd Edition Oxford: Blackwell Publishing
- R.B. Braithwaite, G. Jefferson, Max Newman, and Alan Turing 1952 Can automatic machines be said to think? bbc radio broadcast. In Copeland (2004), pp. 494–506
- Roderick Chisholm 1957 *Perceiving*. Ithaca: Cornell University Press
- Brian Christian 2011 *The Most Human Human: What Talking with Computers Teaches Us About What It Means To Be Alive*. New York: Doubleday
- B. Jack Copeland (ed) 2004 *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life plus The Secrets of Enigma*. Oxford: Clarendon Press
- B. Jack Copeland and Diane Proudfoot 2009 Turing’s test: A philosophical and historical guide. In Epstein et al. (2008), pp. 119–138
- James T. Culbertson 1956 Some uneconomical robots. In Shannon and McCarthy (1956), pp. 99–116
- Donald Davidson 1987 Knowing one’s own mind. *Proc. and Addresses of the Am. Phil. Assoc* **60**, pp. 441–58. (Also in Donald Davidson 2001 *Subjective, Intersubjective, Objective*. New York and Clarendon: Oxford University Press, pp. 15–38.)
- Daniel C. Dennett 1978a *Brainstorms*. Cambridge, Mass.: Bradford Books/MIT Press
- Daniel C. Dennett 1978b Toward a cognitive theory of consciousness. In Dennett (1978a), pp. 149–173. Originally in Savage (1978)
- Daniel C. Dennett 1985 Can machines think? In Shafto (1985), pp. 121–145
- Daniel C. Dennett 1995 *Darwin’s Dangerous Idea: Evolution and the Meanings of Life*. New York: Simon and Schuster
- David L. Dowe and Alan R. Hájek 1997 A computational extension to the Turing test. nil Report 97/322. Department of Computer Science, Monash University
- David L. Dowe and Alan R. Hájek 1998 A non-behavioural, computational extension to the Turing Test. *Proc. Int. Conf. on Computational Intelligence and Multimedia Applications*, pp. 101–106. Gippsland, Australia

- Robert Epstein, Gary Roberts, and Grace Beber 2008 *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*. Springer
- Jerry Fodor 1975 *The Language of Thought*. New York: Thomas Y. Crowell
- Robert M. French 1990 Subcognition and the limits of the Turing Test. *Mind* **99**(393), pp. 53–65. Reprinted in (Shieber 2004), pp. 183–197
- Borko Furht and Armando Escalante (eds) 2010 *Handbook of Cloud Computing*. New York: Springer
- Peter Geach 1957 *Mental Acts*. London: Routledge and Kegan Paul
- Stevan Harnad 1990 The symbol grounding problem. *Physica D* **42**, pp. 335–346
- Stevan Harnad 1991 Other bodies, other minds: A machine incarnation of an old philosophical problem. *Minds and Machines* **1**(1), pp. 43–54
- Stevan Harnad 2000 Minds, machines, and Turing. *J. of Logic, Language and Information* **9**(4), pp. 425–45
- Patrick Hayes and Kenneth Ford 1995 Turing Test considered harmful. *Proc. Ijcai* **14**, pp. 972–977. Vol. 1
- Andrew Hodges 1983 *Alan Turing: The Enigma*. New York: Simon and Schuster
- Owen Holland (ed) 2003 *Machine Consciousness*. Exeter: Imprint Academic
- Steven Homer and Alan L. Selman 2011 *Computability and Complexity Theory*. New York: Springer
- Mark Humphrys 2008 How my program passing the Turing Test. In Epstein et al. (2008), pp. 237–260
- N.D. Jones, C.K. Gomard, and P. Sestoft 1993 Partial evaluation and automatic program generation. With chapters by L.O. Andersen and T. Mogensen. Prentice Hall International
- Timothy Kam 1997 *Synthesis of finite state machines: Functional optimization*. Boston: Kluwer Academic Publishers
- Robert Kirk 1995 How is consciousness possible? In Metzinger (1995), pp. 391–408
- Donald E Knuth 1998 *The Art of Computer Programming: Seminumerical Algorithms*. Reading, MA: AddisonWesley. (3rd edition)
- J.R. Leigh 2006 *Applied digital control: Theory, design and implementation* (second edition). Dover Publications

- Douglas B. Lenat 2009 Building a machine smart enough to pass the Turing Test: Could we, should we, will we? In Epstein et al. (2008), pp. 261–282
- Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley 2012 The Java Virtual Machine specification: Java se 7 edition. (accessed 2012-07-01)<http://docs.oracle.com/javase/specs/jvms/se7/html/index.html>
- Thomas Metzinger (ed) 1995 *Conscious Experience*. Ferdinand Schoningh (English edition published by Imprint Academic)
- P. Millican and A. Clark 1996 The legacy of Alan Turing. Oxford: Clarendon Press
- Donald Perlis 2005 Hawkins on intelligence: fascination and frustration. *Artificial Intelligence* **169**, pp. 184–191
- Richard Purtill 1971 Beating the imitation game. *Mind* **80**(318), pp. 290–94. (Reprinted in Shieber 2004, pp. 165–71.)
- Stuart Russell and Peter Norvig 2010 *Artificial Intelligence: A Modern Approach (3rd edition)*. Prentice Hall
- C. Wade Savage (ed) 1978 *Perception and Cognition: Issues in the Foundation of Psychology, Minn. Studies in the Phil. of Sci.* University of Minnesota Press
- John R. Searle 1980 Minds, brains, and program. *The Behavioral and Brain Sciences* **3**, pp. 417–424
- Michael Shafto (ed) 1985 *How We Know*. San Francisco: Harper & Row
- Claude E. Shannon and John McCarthy (eds) 1956 *Automata Studies*. (Note: *Annals of Mathematics Studies* **34**.) Princeton University Press
- Aaron Sloman and Ron Chrisley 2003 Virtual machines and consciousness. *J. Consciousness Studies* **10**(4–5), pp. 6–45. Reprinted in (Holland 2003), pp. 133–172
- Scott Smith and Jia Di 2009 Designing asynchronous circuits using NULL conventional logic (ncl). Morgan & Claypool Publishers
- John Symons and Paco Calvo (eds) 2009 *The Routledge Companion to Philosophy of Psychology*. London: Routledge
- Alan Turing 1950 Computing machinery and intelligence. *Mind* **49**, pp. 433–460
- Alan Turing and R.A. Brooker 1952 *Programmers' Handbook (2nd Edition) for the Manchester Electronic Computer Mark II.* <http://www.computer50.org/kgill/mark1/progman.html>
- Inco Wegener 1991 The complexity of boolean functions. Wiley

Joseph Weizenbaum 1976 *Computer Power and Human Reason: From Judgment To Calculation*. San Francisco: W. H. Freeman