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

To appear, Minds and Machines. **Visit** `link.springer.com`**.**

Drew McDermott

the date of receipt and acceptance should be inserted later

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, the "humongous table (ht) program," which simply looks up in a table what to say next. This claim is examined in detail. Three ground rules are argued for: 1. That the HT program must be exhaustive, and not be based on some vaguely imagined set of tricks. 2. That the HT program must not be created by some set of sentient beings enacting responses to all possible inputs. 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 HT program could simply be an "optimized" version of some computational model of a mind, created via the automatic application of program-transformation rules (thus satisfying ground rule 2). 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 no one could dismiss out of hand the possibility that the HT program was intelligent. 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.

Keywords Turing test

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[1] 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

Drew McDermott
Yale University
E-mail: drew.mcdermott@yale.edu

[1] 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.

supposed to be. But in general the thought experiment is meant to show that seemingly intelligent behavior 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 [Hayes and Ford(1995)]). But the spell cast by the possibility of the humongous[2]-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 paper "Computing machinery and intelligence" [Turing(1950)] 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* 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[3] 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 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.[4]

[2] 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 be seen 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.

[3] 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 rule will be seen below.

[4] 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 explained in section 2, the judge gets a chance to edit their entries before any part of them is sent to the interlocutor. (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.) Judge inputs that violate constraints such as character limits must be edited until the constraints are satisfied. The two newlines between turns don't count as part of the utterance on either side. We'll always let the judge go first, but they can type the empty string to force the interlocutor to be the first to "speak." The interview ends after an hour or if the judge 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.

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 [Hodges(1983),Millican and Clark(1996)] based on an idea by Claude Shannon [Shannon(1950b),Shannon(1950a)], he would have surely believed there could be 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)Braithwaite, Jefferson, Newman, and Turing] he described the Test in slightly different terms, and went on to say

> My suggestion is just that this [i.e., can a machine 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)Braithwaite, Jefferson, Newman, and Turing, 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 Shannon and McCarthy ([Shannon and McCarthy(1956)]) but the first thorough analysis was that of Ned Block ([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[5] types in sentence A. The machine 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[6] and types out its fourth sentence, and so on (p. 20).[7]

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" (HT program) to refer to the Block algorithm, and introduce names for its variants as they come up. Sometimes

[5] Block's term for what I am calling the "judge."

[6] It's obviously necessary to insert something like the `##` marks because otherwise there would be many possible interchanges that could begin ABC. It's not clear whose turn it is to speak after a conversation beginning "Veni ... Vidi ... Vici. Ave Caesar! A fellow Latin scholar. Great!" Block probably just assumed some such marker would end A, B, and C. I'm making it explicit.

[7] 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.

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 (HT) Argument. 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).[8] It is fair to say that almost everyone who hears the HT Argument agrees with its conclusion.

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.

The HT Argument can be used in two different contexts: (a) Block's original context, as a way of refuting behaviorism; and (b) the context that is my focus, as a way of finding fault with the Turing Test. For Block, it is enough to show that the HT program is logically possible and then invite us to agree that it wouldn't be intelligent; in the context of the TT, however, we care more about prior probabilities. If a bank finds its vaults locked but empty, it is logically possible that the money fell into a random spacetime warp, but Scotland Yard doesn't care about that. They are more interested in which suspects and means are the most probable. The reason why Block's argument seems relevant to context (b) is because many people believe it shows that a practical and "basically trivial" program could pass the Turing Test.

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

1. There must be a fixed time limit for the conversation beween 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 HT Argument. In section 5 I will summarize my conclusions.

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

[8] Shannon and McCarthy [Shannon and McCarthy(1956)] require 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."

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. (p. 184)

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 ([Copeland and Proudfoot(2009)]) seem to assume that the HT Argument 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 [HT Argument] 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 Hájek ([Dowe and Hájek(1997)],[Dowe and Hájek(1998)]) make a proposal for additional requirements on any TT examinee that seems to take the HT Argument 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 HT program 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 judge 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

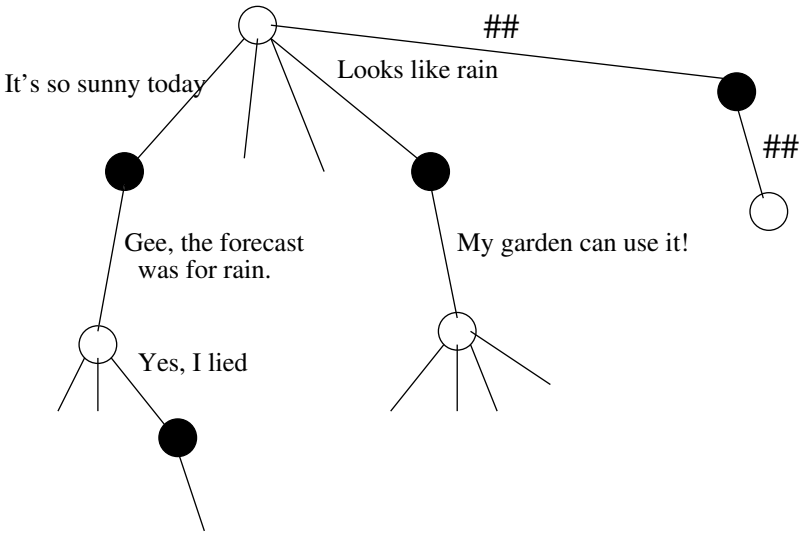

Fig. 1 A few nodes of a strategy tree. White circles represent *J-nodes* where the judge is to speak; black circles represent *C-nodes* where it's the computer's turn.

edge labeled with the string `"It's so sunny today"`. The resulting node (a *C-node*, colored black) has just one edge leading out 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 judge; but after hearing that string, there can again be a single designated response.[9]

I will use the term *strategy tree* for a tree as described in figure 1, for reasons that will become clear. Having described the HT program 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 HT program 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 `##`[10] and the values are strings with no occurrences of `##`. The table might contain the following key-value pair:

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

[9] 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.

[10] Equivalently, odd-length *lists* of strings.

```
                                  ⟶
                  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 HT program 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 HT program 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.[11] 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 HT program that I will

[11] 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.

```
What number is this:
////__xx_____xx
////__xx_____xx
////__xx_____xx
////__xxxxxxxxx
////_________xx
////_________xx
////_________xx
?
```

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

discuss. I will use the term *timed string* to refer to these ⟨string, timing-data⟩ pairs.[12] I will leave modification of Block's sensible-string table to include timing data as an exercise.[13]

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.

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. We'll do that by converting newlines ("carriage returns") to single spaces. However, nothing forbids the judge from asking, "What number is this when each sequence of slashes is converted to a newline:. . . ?" So in addition we will 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 to be disposed of, and that's how to deal with timing the *judge's* inputs. Suppose the judge could type

```
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 quicker, the first or the second?
```

[12] If we want to allow interlocutors to edit lines before they are seen by the judge, then times should be associated with completed lines, not individual characters. If we really want to avoid reaction times completely, then we can introduce random delays (as we do for the judge; see p. 8) or 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.

[13] One more restriction: Timed strings can't have times so short that the typing speed exceeds the rate at which 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 set of all possible examinees, as we will in section 4.2, it's necessary to set bounds on their abilities to keep the set finite.

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

To sidestep this problem, we'll assume that the judge's input is collected in its entirety and then fed to the interlocutor as a chunk. If it's too long or violates other constraints (see section 3.1), the judge will have to edit it until it's legal. An additional random delay could be thrown in to baffle all attempts at figuring out how fast the judge typed.[14]

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. In game theory the payoff is given, but in the TT it is decided by the judge 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.[15]

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 HT program.

> . . . [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.[16][These would be the edge labels below the top node] of the [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.]

[14] We could do the same with the interlocutor's output, but it's traditional to put the burden of replicating human timing and error patterns on the examinees.

[15] 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.

[16] 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 "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 [Harnad(1991)] calls the Total Turing Test (TTT):

> 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].[17]

Adapting the HT program 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 HT program is truly humongous, even in the ordinary Turing scenario, but the HT program 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.[18] 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 judge challenges the robot to a game of ping-pong, the model of the judge'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?

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,

[17] 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.

[18] 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.

"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.[19] But the robots in the TTT must know they're pretending to be human. Otherwise they might wonder why their pudenda are neither concave nor convex; 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 noticing 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 deliberatedly 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 judge." In fact, in spite of some early success with naive subjects, reportedly anecdotally by [Weizenbaum(1976)] and [Humphrys(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)Epstein, Roberts, and Beber] are disappointly 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 next paragraph) to be able to react more flexibly with fewer rules than the HT program. 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

[19] Of course, a truly intelligent examinee 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 HT program 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?

of anything, and no ability to grasp the topic or maintain the thread of the conversation. They are ridiculously easy to expose.

Let me explain the *register* trick, a key example of what Ground Rule 1 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 ourselves 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."

If we use this template in place of a sentence in the lookup table, then the variable `?R` will match any name ("Pradeep" or "Michio" or whatever), and the name will be 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 HT program 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 the strategy tree is that 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 HT program 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" $95^{500} \approx 7.35 \times 10^{988}$ ASCII strings of length 500. But all but a tiny fraction of these strings are complete gibberish.[20] 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:[21]

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.
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.

A fairly careful estimate[22] of the number of inputs these constraints allow is 3.67×10^{445}. This looks like a considerable improvement on 7.3×10^{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 2×10^{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 $(3.67 \times 10^{445})^{50}$, or about $10^{22,278}$ 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 $10^{22,278}$, 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.

[20] Jorge Luis Borges's vision [Borges(2000)] of a library of all possible books of a certain size conveys the idea.

[21] For the exact rules, see appendix B.

[22] See appendix B

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.

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 HT program 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 $10^{22,278}$ 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[23] 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 a TT examinee must be a stand-alone device, and not a mere conduit for transmission of messages from entities stipulated in advance to possess mental states, which is precisely what our little mega-teams of tiny actors are. 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.

> 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

[23] And perhaps a cognitive psychologist.

> 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 ([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 [judge] 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 [judge] 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^{445} possible inputs to the program. In the second scenario in our sequence we hire 10^{445} (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^{445} 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^{445} 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 $(3.67 \times 10^{445})^2 \approx 10^{891}$ actors, each of whom is given a two-input sequence to think about. More precisely, we first use 10^{445} actors to record a response for each potential input. Then, we use (about) 10^{891} 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^{445} times as fast as a real person, we would need only 10^{445} 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 $(3.67 \times 10^{445})^3 \approx 5 \times 10^{1336}$ 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 $10^{22,278}$ 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.

Of course, Block merely wanted to establish that an HT program was logically possible. He could now say that there are many possible strategy trees encapsulating seemingly intelligent conversationalists, and all we have to do is imagine one being embodied. Perhaps the object encoding the tree came 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 processses at work in a swamp [Davidson(1987)].)

But 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-lookup program that answers all yes/no mathematical questions. That is, the judge types in a conjecture, and this HT program 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.[24] Although the set of all possible tables includes tables that perform amazing feats of creation,[25] all but an infinitesimal fraction of them just print (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 $10^{22,278}$ 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.[26] 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 HT program 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 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.

[24] I allude once again to "The Library of Babel."

[25] 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.

[26] 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). How such a huge thing is to be rebuilt "often" is not clear.

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 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 any 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, and 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.

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. Implementing this theory in silion 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 $C_x(s)$.[27] Whether M can experience what x would or remember experiencing 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 HT program, 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 behav-

[27] 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.

ior 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.[28]

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 $y_0 = C_{\mathrm{AB}}(s_0)$, and the two receive the same typed inputs from the judge, 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 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 $TS_{\mathrm{AB}}(s,I)$ 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 $TS_{\mathrm{AB}}(s,I)$, 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 $s \in TS_x(s_0,I)$.

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 the current judge 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)Braithwaite, Jefferson, Newman, and Turing]). 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 the judge being able to detect that fact.

4 The Refutation

We are finally ready to work on two arguments that refute Block's claim that the HT program "all the intelligence of a toaster." Both of them come down to this: The state space for any

[28] 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 judge might be able to figure out who's who by asking the participants to describe the location where they're sitting.

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 Possibility of HTPSs 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 HT program are of an 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.[29] 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^{445} "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 ε if we had to) and merely accepts the next character. Only when the end of the string is reached does a transition to a C-node occur (a transition labeled with `##`). 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. The behavior diagrammed in figure 3 omits any editing the judge performs on their input; the diagram just shows what happens as characters are read from the edited buffer.

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 next output. Symbolically, these components implement a function T with domain and range described thus:

$$T : S \times I \rightarrow S \times O$$

[29] When a leaf state is reached, the FSM halts.

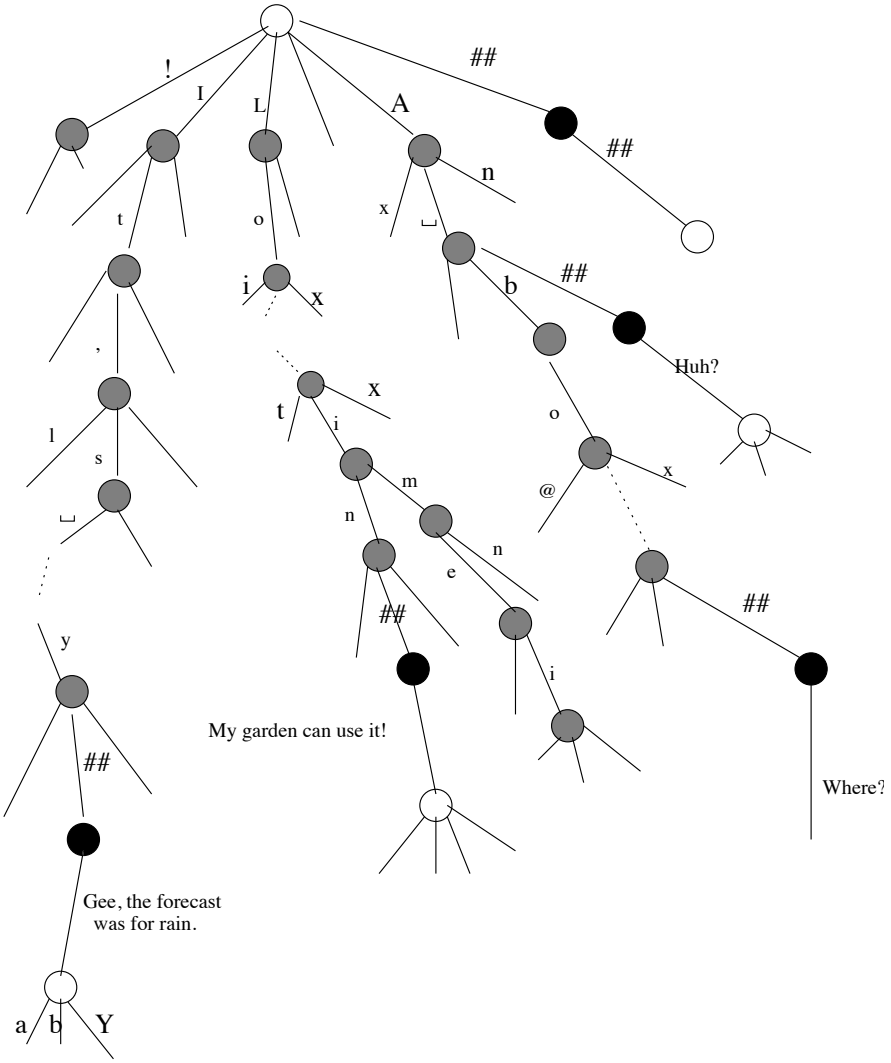

Fig. 3 An alternative version of the strategy tree of figure 1. Gray circles represent intermediate (A-) nodes between J-nodes and C-nodes.

where S is the set of states, I is the set of possible input strings, and O is the set of possible output strings.[30] 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

[30] 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.

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 $10^{22,278}$ (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 70,000 — about 9 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.[31] 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 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.[32] 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 3.3, 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 a judge.

[31] 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.

[32] Using multiple tapes is a convenient device that doesn't change the computational power of Turing machines [Homer and Selman(2011), ch. 2].

We now have a "finite Turing machine" (FTM). Its state space is the state space of the original machine × the possible configurations of the four tapes, including their tape-head positions. 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 its 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, except for being finite, but this is a constraint imposed by the bound on the length of the conversation.

Hence huge HTPSs 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 Possibility of HTPLs Proves Nothing.

However, this argument leaves the possibility that a *particular* instantiation of HTPS is trivial, namely, an HTPL. Braddon-Mitchell and Jackson force us to focus on HTPLs when they phrase their critique as the claim that the HT program "... 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 a *person* would have if they did the job the computer does, or bemoaning the absence of those mental states.[33] It seems to me that simply identifying the unsupported anthropomorphism is enough to defeat it, but if that doesn't convince you, then 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)Lindholm, Yellin, Bracha, and Buckley]

[33] Another example is Searle's ([Searle(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 into," and so on.

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.[34]

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 judge's inputs separated by occurrences of `##`.[35]

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 $C_{\mathrm{AB}}(s)$, 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 that 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.[36] 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,[37] 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)
```

[34] 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?

[35] Recall that in section 2 we "optimized" keys by removing the examinee's contributions to the dialogue.

[36] Of course, some people contend that it is absurd to deny a creature phenomenal consciousness if *it* doesn't seem to believe it lacks anything [Dennett(1978), McDermott(2001)].

[37] 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.

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.[38] 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 judge. So we can idealize the program as consisting of one outer loop, during every iteration of which "AB" reads a query from the judge 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
        {
          KB_new ← react to R using KB;
          aunt_bertha_loop(KB_new)
        }
   }
```

Here KB is the program's knowledge base.[39] 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 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):

[38] A set of deterministic processors acting asychronously 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.

[39] It may seem unusual to compute a new knowledge base rather than make changes to the old one, but it's 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.

```
  {
    Settle down and initialize;
    aunt_bertha_loop(KB_0)
  }
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 appendix C, 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 appendix C 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 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 be lose their faculties when their time is up.[40]

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 deterministic 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 HT program *must* be restarted, because unbeknownst to it its lifetime is very short.

However, there is no particular reason to make the HTPL deterministic; as long as we're implementing a tree of size $10^{22,278}$, we might as well multiply it by another factor of 10^6 and create 1,000,000 different slightly different Aunt Berthas by retaining some of the randomness in the full computational model C_{AB}. If there are 10 judges, there is only one chance in 10,000 of two of them having the same conversation, assuming they all conspire to say the same things in the same context.

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

[40] One might object that a person sentenced to capital punishment could always get a last-minute reprieve from the governor; their hopes and dreams are never necessarily futile. So imagine someone poisoned by an irreversible infusion of nanobots that snip out pieces of brain one by one until after an hour the victim is dead.

it weren't for the bizarre economic constraints in the fanciful universe of the HT Argument, which are obviously quite different from the 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 HT Argument 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 HT program 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 purely formal operations, which can be carried out by a simple algorithm. M_{AB} has mental states, but the transformations that take it to HTPL do not. Although the operations described in appendix C may be applied in more than one order, applying one of them doesn't block another, so the destination is always the same except for the names of variables. Of course, we still have the problem of implementing this transformation algorithm, and running it would take many eons; but all this is just as conceivable as the HT program itself.

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 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 `if`s operates under. Only a finite number of things can happen to M_{AB} because, even though most of the time $10^{22,278}$ 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.[41] Hence we *can* produce a finite analysis of every mental-state term that applies to "Aunt Bertha" as 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 HT program 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 [French(1990)]. If we restrict our attention to the vanishingly small subset of HT programs that *can* pass it, how

[41] Of course, she can *discuss* them, and probably will if the judge brings them up.

can we possibly dismiss the possibility that a candidate instance is a hyperoptimized version of a respectable mental model?

Given our ideas on what it means for an HT program 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.

In section 3.3 I warned you I would use the word "remembers" in contexts like these to mean "behaves for all the world as if it remembers." That is, the HT program is able to talk about the event it remembers, and in some versions this talk is mediated by data structures that look like what we might traditionally think of as data structures of the right type (neural-net weights, predicate-calculus formulas, or whatever). Only in HTPL is the state boiled down to the string of sentences the judge has typed.[42] By these criteria, the HT program is never in the second state listed above; it will never remember looking any of its responses up, even if some looking up occurred within it. This is hardly surprising; *you* have no memories of transmitting signals from your hippocampus to your frontal lobe.

To recapitulate:

- (Section 4.1) An HTPS is essentially a description of a finite-state machine, but exactly what computations are performed in the transition from one state to another is not constrained much by the finiteness requirement. Over a one-hour time horizon, the computations could be isomorphic to those performed by an intelligent program with an indefinite lifespan.
- (Section 4.2) An HTPL is a much more specific and constraining description, but even so there is no way to rule out the hypothesis that it is a version of an intelligent program "optimized" by the application of formal transformation rules, under the assumptions that it has a one-hour lifespan and that a universe-filling chunk of storage space costs absolutely nothing.

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 that 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 HT program.

My argument depends crucially on my ground rule 3, the premise that cognitive science is onto something, and that computational models could turn out to have some validity. If you reject this premise, then neither of Block's contenders, behaviorism or psychologism, will have much appeal.

At the end of section 1, I distinguished two uses to which the HT Argument has been put: (a, Block's original purpose) As a way of refuting a behavioristic definition of "conversational intelligence"; (b) as a way of dismissing the Turing Test as a sufficient test for

[42] So the state of remembering the name of the judge is mediated by the disjunctive state consisting of all string sequences in which the judge tells AB their name and AB is able to recite it correctly later.

intelligence. The arguments of Section 4 are in the context of use (a); they grant the logical possibility of the HT program and refute the conclusion that such a thing lacks intelligence.

Along the way to those arguments, in section 2, I pointed out many obstacles toward granting the assumption that physical embodiments of the HT program could, in principle, be built. Although I didn't argue the issue in detail, I believe the extension of this assumption to the control of robots, an extension made fairly casually by both Block [Block(1981)] and Braddon-Mitchell and Jackson [Braddon-Mitchell and Jackson(2007)], is unwarranted (section 2), even if the original assumption is true. But is it? It's never seemed plausible to me that something so huge could be built and kept up to date. In this paper I outline two more strikes against Block's scenario:

1. If the table is constructed by having a large but tiny improv troupe enact all possible one-hour conversations, then the table is essentially a communication channel between them and the judge. The HT program would, under that picture, not be a self-contained automaton, and not relevant to either use (a) or use (b) of the HT Argument.
2. An object of the size of the HT program, even if it could be crammed into some kind of hyperspace warp, even if humans and their tools could be shrunk to the size of the builders of the humongous table, could never be examined in any detail. Everything about it, including its provenance, would have to be taken on faith.

The second obstacle is irrelevant to use (a); if we're asked to *stipulate* that something is true, it does not matter whether we can know whether it is true. But the Turing Test is, after all, a *test.* If we were forbidden from examining the insides of a successful entrant in a Turing competition, which hypothesis has the higher prior probability: that it embodies advances in AI, computational linguistics, computational neuroscience, and machine learning; or that it contains a hyperspace conduit to an infinitely divisible region of spacetime?

The difference between use (a) and use (b) has been insufficiently appreciated in the past. The reason for this, I'm guessing, is that intuitions about how big the humongous table would have to be to pass the TT, even if it were enhanced with a few tricks as discussed in section 3.1, have been insufficiently educated. Even Block can't resist wondering whether "[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 if the strategy tree embodied by the HT program has $10^{22,278}$ nodes, this speculation is futile. That fact ought to force everyone to decide whether it's pure logical possibility they care about, or some application to the Turing Test.

It may seem that I have used a sledge hammer to smash a flea, that the HT Argument is too peripheral to 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 and does nothing else fails to be truly intelligent, so of course the humongous-table program does too. But the HT Argument 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 psychologistic 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 a credible Turing-test contestant, and just couldn't be intelligent. I hope I have at least gotten some doubt to creep into your mind regarding these intuitions.

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.

Acknowledgements Thanks to Dana Angluin, Ned Block, Matt Ginsberg and especially the anonymous referees for useful suggestions. I take responsibility for the flaws that remain.

A A Simple Programming Language

The algorithms used in appendices B and C 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

```
{
  s1 ; s2 ; ... ; sl
}
```

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 v1 = e1
    v2 = e2
    ...
    vk = ek
   in
     e0
```

which evaluates $e_1, \ldots, e_k$, then binds $v_1, \ldots, v_k$ to the resulting values during the evaluation of e_0.

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

Pseudo-code is written with italics.

B Estimation of Number of Possible Inputs

In this appendix I describe how the estimate of 10^{445} possible strings the judge can type was arrived at. Things will quickly become unclear if I don't distinguish consistently between character tokens (occurrences) and character types; from now on when I do not explicitly use the word "type" I am referring to tokens. I'll use *symbol* as a synonym for *character type* to avoid tedium. Also, I'll use British conventions for punctuation after quotes, and put it outside the quotation marks. It's more logical that way, if not as attractive.

Let's go over the rules again for what the judge can input. They are allowed a maximum sentence length of $w \approx 85$ words and $h \approx 500$ characters, including punctuation. All nonprinting characters are treated as whitespace, and a group of them is converted to a single whitespace.

Let's start with punctuation, which comprises the characters

```
"‘’$%,.:;!?@()[]{}-/|\
```

and which is to be used for two purposes: to tie two words together or to come at the beginning or end of a word. (In other words, a group of punctuation characters is not bounded on both sides by whitespace; but see below.) Let P be the number of punctuation symbols.

A *word* is a sequence of non-whitespace characters generated in one of two ways:

1. Most words are presumed to be drawn from a list of the C most common words in English text. However, our model of a word is extremely restrictive. We count different forms of a word as different words, and the case of a word's characters matter. So "Fool," "fool" "fools," "fooled," and "fooling" are all counted as separate words. This may seem ungenerous, but assuming all forms to be equally likely will only make the table larger, and I'm trying to make it as small as possible.
2. Because we must allow for many words not in the dictionary, we give the judge a certain number of "arbitrary characters." The number depends on the number L_c of occurrences of common words in the sentence. The judge gets $\lfloor r_a L_c \rfloor$ arbitrary characters, where r_a is a constant (say, around 5); but they get at least $a_{\min}$ and at most $a_{\max}$ arbitrary characters. Let's call this function $N_a(L_c)$; the important thing about it is that is nondecreasing. There is a similar function $N_p(L)$ that specifies the number of punctuation symbols the judge is allowed, but as a function of the total number of words, not just the common ones.

An *uncommon* or *strange* word is a nonempty sequence of arbitrary characters. They are drawn from the alphanumeric character types, plus the symbols in the following list:

```
#&*+/<=>^_~
```

There are a total of A arbitrary character types. No symbol is classified as both a punctuation character type and an arbitrary character type. This may seem surprising, because we expect judges to type sentences like these:

```
Franklin's brain --- trust me --- was not as good as Eleanor's
Franklin's brain-trust was not as good as Eleanor's.
```

In the first, "`---`" might be an uncommon word. In the second, the hyphen is a word-tying punctuation symbol.

We manage to allow for punctuation and arbitrary symbols to be disjoint with two ad-hoc rules:

– At the beginning of a sentence a freestanding punctuation group is treated as attached to the first word. In all other sentence contexts it is attached to the word on its left. So the first sentence of the pair above will be seen by the interlocutor as

  ```
  Franklin's brain--- trust me--- was not as good as Eleanor's
  ```

  In either case it doesn't matter how many punctuation characters are already attached to the word in question; extra punctuation is just jammed on. So

  ```
  Franklin's ‘‘brain’’? ! ; trust me --- it wasn't as good as Eleanor's
  ```

  is treated the same as

  ```
  Franklin's ‘‘brain’’?!; trust me--- it wasn't as good as Eleanor's
  ```

– We allow the judge to type a series of punctuation symbols and nothing else, such as "`!!!`". It can't be longer than $N_p(0)$, though.

If we allow punctuation and arbitrary character types to overlap, it will be hard to avoid counting many strings twice. If hyphen is in both groups, then we have to worry about examples such as "`A-I`", which can be considered to be an uncommon word, or two common words tied together by a piece of punctuation.

None of this is written in stone. It just gives you an idea of the magnitude of A, the number of arbitrary-character types; and P, the number of punctuation symbols.

What value shall we use for C, the number of commonly used words? The English language grows constantly, but there is a core set of words said to be of size $\approx 250,000$.[43] The distribution of words obey's Zipf's Law, which states[44] that the frequency of a word is inversely proportional to its rank (its position in a list of words sorted in decreasing frequency order).

$$P(\text{word}) \approx \frac{1}{r \ln(1.78R)}$$

[43] Source: `http://oxforddictionaries.com/us/words/-how-many-words-are-there-in-the--english-language`.

[44] Source: `http://mathworld.wolfram.com/ZipfsLaw.html`.

Values of parameters			
h	500	$a_{\min}$	10
w	85	$a_{\max}$	40
C	10,000	r_a	5.0
A	72	$p_{\min}$	3
P	22	$p_{\max}$	10
l_{typ}	5	r_p	0.5
$l_{\max}$	15		

Table 1 Default values for `TuringCombos` parameters.

where r is its rank in the distribution and $R \approx 250,000$ is the size of the language. So suppose we want to know how many words we would need to include in our list so that it contained 90% of the words in common use. That's the $r_{0.9}$ such that

$$\sum_{i=1}^{r_{0.9}} P(\text{word}_i) = 0.9$$

A quick back-of-the-computer calculation yields $r_{0.9} \approx 60,000$.

The problem with this estimate for C is that it's dealing with *words,* presumably drawn from the set of all written works in English. On the one hand, it neglects the fact that we actually want the corresponding estimate for *word forms*. On the other hand, the set of words used in Turing tests is presumably not that similar to the set used in the average Saul Bellow novel.

There is no substitute for real data, so we turn to the "Enron e-mail" dataset, preserved as part of a DARPA research project[Leber(2013)]. A nice clean version of a subset of these messages has been prepared by Will Styler at the University of Colorado [Styler(2011)]. It consists of all messages sent by Enron employees over some period of time. (Dates and other identifying and potentially embarrassing information has been removed.) Although headers and quoted messages have been deleted, there is still some junk that one needs to ignore, especially some odd XML tags, but it's not hard to harvest all 14,470,491 distinct word forms.[45] Surprisingly (to me), 6631 of them account for 90% of all occurrences. They do *not* seem to obey Zipf's law, although perhaps that's because "the" (559291 occurrences), "The" (53544), and "THE" (2484) are counted as distinct word forms.

So what value should we use for C? I've speculated all over the place, but finally wound up where I began, at 10,000, based on the Enron data. At the end of this appendix I discuss the sensitivity of this model to changes in parameters.

Table 1 shows the parameters we have so far. The parameters on the right define $N_a(l)$ and $N_p(l)$, the bounds on arbitrary and punctuation characters. Those on the left give us the size of sentences, alphabets, and the dictionary. The first parameter, h, is the bound on the overall number of characters the judge types; w is the bound on the number of words, both common and not so common. The quantity l_{typ} is the "typical" length of a word in the dictionary; $l_{\max}$ is the length of the longest word. I'll explain the need for these parameters later.

Before proceeding further we need a few combinatoric functions, besides the familiar $n!$ and $\binom{m}{n}$, the number of combinations of m things taken n at a time.

The number of ways of distributing a sequence of l objects into n ordered bins (possibly putting 0 in some bins) is

$$N_{\text{distrib}} = \begin{cases} 1 & \text{if } l = 0 \\ 0 & \text{if } l > 0 \text{ and } n = 0 \\ \binom{l+n-1}{l} & \text{if } l > 0 \text{ and } n > 0 \end{cases}$$

The number of ways of *filling* a sequence of n bins with a sequence of l objects — that is, leaving none of the n empty — is

$$N_{\text{fill}} = \begin{cases} 1 & \text{if } n = l = 0 \\ 0 & \text{if } l > 0 \text{ and } n = 0 \\ 0 & \text{if } 0 < n \text{ and } n > l \\ \binom{l-1}{n-1} & \text{if } 0 < n \leq l \end{cases}$$

[45] Some of them are numbers and deliberate nonsense; but some common words are numbers, too.

The number of ways of merging two sequences of objects, one of length l_1 and the other of length l_2, dividing the former into d pieces, is

$$N_{\text{merge}}(l_1, l_2, d) = \binom{l_1 - 1}{d - 1}\binom{l_2 + 1}{d}$$

The number of ways of interleaving two sequences of length l_1 and l_2 is then

$$N_{\text{interleave}}(l_1, l_2) = \begin{cases} 1 & \text{if } l_1 = 0 \text{ or } l_2 = 0 \\ \sum_{d=1}^{l_1} N_{\text{merge}}(l_1, l_2, d) & \text{if } l_1 > 0 \text{ and } l_2 > 0 \end{cases}$$

After all that, we can proceed. The estimate of the total number of possible inputs is

$$(\texttt{if } w > 0 \texttt{ then nPuncGroups() else 0}) + \sum_{l=0}^{w} \texttt{nWordSeq}(l)$$

where `nPuncGroups()` is the number of possible freestanding punctuation groups (see p. 31) and `nWordSeq`(l) is the number of word sequences of length l.

The first is simply disposed of:

$$\texttt{nPuncGroups()} = \sum_{n=1}^{p_{\min}} P^n$$

That gets us to `nWordSeq`. At some point we have to iterate through the number of arbitrary and punctuation characters used in a sequence, and we choose to do this early, because otherwise it will be difficult to impose the requirement that the total number of characters be $\leq h$ (about 500).

So `nWordSeq` is defined as follows:

```
define nWordSeq(len)
   if len = 0 then 1
   else
      let arbCharLim = Na(len − 1) in
        arbCharLim
           Σ   nWordSeqA(len,a)
          a=0
```

where `nWordSeqA`(len, a) is the number of word sequences of length l built using a "arbitrary" characters.

```
define nWordSeqA(len, a)
   let puncCharLim = Np(len) in
     puncCharLim
        Σ    nWordSeqAP(len, a, p) * nWaysPunctuate(len, p)
       p=0
```

I'll come back to `nWaysPunctuate(len, p)`, the number of ways to punctuate a word sequence of length `len` using exactly `p` punctuation symbols. The focus will be on `nWordSeqAP`.

In what follows I will refer once or twice to the "inverses" of N_a and N_p. Recall that $N_a(w)$ was defined as the maximum number of arbitrary characters that can be added to a sequence of w common words, that is, used to make uncommon words. Although $N_a(w)$ is not in general invertible, it is nondecreasing, so we interpret $N_a^{-1}(c)$ to be the *minimum* number of common words required to justify the presence of c arbitrary characters. (If $c > a_{\max}$, then $N_1^{-1}(c) = \infty$.) Similarly, *mutatis mutandis,* for $N_p^{-1}(c)$.

Here's the definition of `nWordSeqAP`:

```
define nWordSeqAP(len, a, p)
   if a=0
     then nComWordSeq(len, h-p)
     else let minComWords = max(len - a, Na⁻¹(a))
              maxComWords = if C = 0 then 0 else len-1
          in
              maxComWords
                  Σ        (nCommonWordSeq(c, h-a-p)
             c=minComWords   * nStrangeWordSeq(len-c, a)
                             * Ninterleave(c, len-c))
```

`nCommmonWordSeq`(n,l), the number of common-word sequences of length n using no more than l characters, can be computed as the sum of the number of common-word sequences of length n using 0 characters, the number using 1 character, and so forth, up to the number of sequences using exactly l characters.

$$\texttt{nCommonWordSeq}(n,l) = \sum_{c=0}^{l} \texttt{nCommonWordSeqExact}(n,c)$$

This may seem like a bizarre direction to go in, but bear with me.

`nCommonWordSeqExact` obeys the following recursion

$$\texttt{nCommonWordSeqExact}(n,c) = \begin{cases} \sum_{d=1}^{\min(l_{\max},c)} (\texttt{if } d = c-1 & \text{if } n>0 \text{ and } c>0 \\ \quad \texttt{then } 0 & \\ \quad \texttt{else numCommon}_d & \\ \qquad \texttt{* nCommonWordSeqExact}(n-1,r) & \\ \quad \textit{where } r = (\texttt{if } d = c \texttt{ then } 0 & \\ \qquad \texttt{else } c-d-1)) & \\ 1 & \text{if } n=0 \text{ and } c=0 \\ 0 & \text{otherwise} \end{cases}$$

Here $\texttt{numCommon}_d$ is the number of common words with d characters. This number is precomputed from whatever corpus we use to estimate our parameters. The recursion relates all the word lengths from 1 to $l_{\max}$ to shorter sequences (length r) obtained by snipping off $\texttt{numCommon}_d$ characters, plus 1 for the interword space (which is not necessary if no characters remain after subtracting d). If $\texttt{numCommon}_d$ is multiplied by the number of sequences of $n-1$ common words and r characters, we get the number of sequences of n common words and c characters starting with a word of d characters.

If one were to implement this recursion directly, the complexity of the resulting algorithm would be exponential. Fortunately, fixing this problem is a simple exercise in dynamic programming [Bertsekas(1987)]. We "memoize" the function `nCommonWordSeqExact`, meaning that we create an array big enough to hold all the values and cache them there as they are computed. ("Big enough" $= w \times h$, i.e., 85×500.) The procedure `nCommonWordSeqExact` checks the array and simply returns the value stored there, if any; otherwise, it computes the sum and stores it in the array.

Finally, we explain `nWaysPunctuate`(l,p), the number of ways of punctuating a sequence of l words using exactly p punctuation characters. We allocate a certain number of punctuation characters to tying words together, and a certain number to putting at either end of the resulting "superwords."

$$\texttt{nWaysPunctuate}(l,p) = \begin{cases} 1 & \text{if } l = 0 \\ & \textit{otherwise} \\ \sum_{k=0}^{\min(l-1,p)} \binom{l-1}{k} \texttt{nWaysPuncG}(l-k,k,p) & \end{cases}$$

The first factor being summed — $\binom{l-1}{k}$ — is the number of ways of choosing k glue points. The second, `nWaysPuncG`(l,g,p), is the number of ways of choosing p punctuation characters so as to fill g glue points and decorate the resulting l superwords.

```
define nWaysPuncG(l,g,p)
   let m = (if g=0 then 0 else p) in
```

$$P^p \sum_{n=\mathtt{g}}^{\mathtt{m}} N_{\text{fill}}(n,\mathtt{g}) N_{\text{distrib}}(\mathtt{p}-n, \mathtt{2*l})$$

The last line may be parsed thus: There are P^p sequences of p punctuation characters. At least g must be allocated to fill the g glue points, but as many as p can be allocated for that purpose. If n is the number allocated to gluing, then there are $N_{\text{fill}}(n,g)$ ways to do that, and $N_{\text{distrib}}(p-n,2l)$ ways to sprinkle the remaining punctuation characters around the fronts and backs of the l superwords.

I have now laid out the entire algorithm.[46] In spite of all the ingenuity invested in it, there is still a problem with `nWordSeqAP` (p. 33). It will overcount in situations like the following. Suppose `len`=10 and

[46] There are many missing details, of course; if you are really obsessive, the Java version of all these procedures is available. See below.

a=20. $N_a(3) = 15$ and $N_a(4) = 20$, so $N_a^{-1}(20) = 4$. The algorithm considers c in the range $[4,9]$. When c=5, 5 slots are left for strange words. The average length of the common "words" in the Enron corpus is about 5 characters.[47] If 5 strange characters are placed in each of 3 slots, and the remaining 5 characters are sprinkled among the remaining 2 slots, then most sequences of three common English words will be included in the sequence of 5-character strange words.

In general, I will write $N_t(a,w)$ to denote the number of words of typical length that can be created using a characters, under the constraint that a total of w "words" have to be created. (Above, we were considering $N_t(20,5)$.) The function is defined thus:[48]

$$N_t(a,w) = \begin{cases} \lfloor \frac{a-w}{l_{\mathrm{typ}}-1} \rfloor & \text{if } a \leq w l_{\mathtt{typ}} \\ w-1 & \text{otherwise} \end{cases}$$

Rationale: You have to fill all the w "word bins" with one character at least, which leaves $a-w$ characters to be distributed. But now we only have to get $l_{typ}-1$ characters in each bin to hit the target. However, if there are too many characters the best we can do is put l_{typ} in $w-1$ bins and the remainder in the last bin, which will contain one long, probably bizarre word.

On a previous iteration of nWordSeq with len=10, when a was 15, all the common-word sequences of length 6 would be counted, combined with uncommon-word sequences filling 4 bins. $N_t(15,4) = 2$, which means two long strange words and a few characters left over. I submit that most of these are overcounts, because when we get to the case $a = 20, c = 5$ (which this example started with, remember?), we're going to generate all the sequences of 6 common words plus two strange words plus 3 more characters, plus a lot more.

Fortunately, it's easy to state and check when a state of the count will be made redundant by a later state. Writing a state as a triple

$$\langle \text{common-word-num, arb-char-num, total length} \rangle$$

A triple $\langle c,a,l \rangle$ is *subsumed by* $\langle c-1, a+l_{typ}, l \rangle$ if and only if (letting $a' = a+l_{typ}$):

1. $N_a(c-1) \geq a'$ (or we can't justify so many arbitrary characters)
2. $N_t(a', l-c+1) = N_t(a, l-c)+1$ (so we get one more strange word of the right length)

The implemented version of nWordSeqAP incoporates this check: subsumed count states are skipped.

In spite of all these cogitations, the effect of the overcount turns out to be negligible. My current best estimate of the number of possible inputs a judge could input is

$$3.67235698473639353 \times 10^{445}$$

With the parameter avoidCommonWordOvercount set to false, leaving everything else the same, the estimate becomes:

$$3.67235699290100972 \times 10^{445}$$

which differs only by an amount equal to 2.2×10^{-9} of the original estimate.

It's not really surprising that this source of overcounting is so negligible. By setting the parameter signifTerms to K, we can tell the system to ignore all but the K largest terms of every single sum of increasing terms that it computes. (Its default value is -1, which means that all terms are computed.) If signifTerms=2, the estimate is $3.67235699290100972 \times 10^{445}$, which differs only by an amount $= 3.6 \times 10^{-4}$ of our estimate. It's reassuring that we could have been a lot cruder and still hit our target. If we take the 3 most significant terms of every increasing sum, the difference drops to 6×10^{-6}.

In other words, if we think only about inputs involving 83, 84, or 85 words, and 38, 39, or 40 arbitrary characters, we get a result essentially identical to the full-blooded version. The reason avoiding common-word overcount buys us so little is that it affects mainly smaller terms. The only comfort we can take is that, as far as I can tell, every possible source of overcounting has been considered.

A little sensitivity analysis:

– If C is made 10 times larger $(100,000)$, the effect is fairly dramatic. The estimate rises to 1.51×10^{529}.

[47] Rothschild [Rothschild(1986)] found a value of 7 characters for the most common word length in English. But he was searching for words in novels and other written sources, not harvesting "word forms" from an e-mail archive.

[48] Thanks to Dana Angluin for valuable suggestions here.

- Similar effects occur if we allow more arbitrary characters. Setting $a_{\min}$ to 20 and $a_{\max}$ to 80 produces an estimate of 7.33×10^{513}. There's not much point in experimenting with values for $a_{\max}$ 10 times larger than the range I'm allowing; that just increases the amount of the gibberish we've been trying to reduce.
- Going the other direction, setting $a_{\min} = 5$, $a_{\max} = 20$, $r_a = 3$, shrinks the space to 6.46×10^{410}.
- Of course, we can ultimately make the space as small as we like by shrinking the parameters. But even if we set h to 140, the size of a tweet (and w to 25), the space is still of size 1.06×10^{189}, or 5,000 times the size of the visible universe in cubic Planck lengths.

Anyone interested in experimenting with this program, whose name is `TuringCombos`, can find it at `http://www.cs.yale.edu/homes/dvm/software/turing-combos.tar.gz`. It's written in Java, and instrumented so one can watch how the numbers add up. All the parameters mentioned here can be changed by creating a file with the desired settings and passing it to the program as a command-line argument.

C 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 procedure's operations in advance when some of its arguments are known [Jones et al(1993)Jones, Gomard, and Sestoft].

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. (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 evalution 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 = (read some source ...) in
    A[r]
```

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

```
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.[49]

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

[49] 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 treat randomness elimination as a special case of input anticipation.

```
let v1 = e1
    v2 = e2
    ...
    vk = ek
    in
      e[v1, ..., vk]
```

may be transformed into

```
u1 ← e1
u2 ← e2
...
uk ← ek
e[u1, ..., uk]
```

where each u_i is a variable that occurs nowhere else in the program and $e[u_1, \ldots, u_k]$ is e with every free occurrence of v_i replaced by u_i. (The new variables have global scope.)

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 Pk then
      { ck ;
        if Pk1 then { dk1 }
        ...
        else if Pk,kk then {dk,kk }
      }
```

may be transformed into

```
{
  if P1 then c1
  else if P2 then c2
  ...
  else if Pk then ck ;
  if P1 and P11 then { d11 }
  else if P1 and P12 then { d12 }
  ... else if P1 and P1,k1 then { d1,k1 }
  else if P2 and P21 then { d21 }
  ...
  else if Pk and Pk1 then { dk1 }
  ...
  else if Pk and Pk,kk then {dk,kk }
}
```

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`. The "*react*" pseudocode has been altered to return the total time T_C it took to type the output, a timed string; 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, KB0)
```

which is equivalent to

```
{
 let R = (read next query from judge) in
     {
      ⟨KB_new, T_C⟩ ← react to R using KB0;
      aunt_bertha_loop_t(1 hour−T_MJ(R)−T_C, KB_new)
     }
}
```

(We do one function unfold, then one constant fold; 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^{445} 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^{445}-way branch[50] after the input statement. The program has become the version shown in figure 4.

Within each branch in figure 4, the values of both `R` and 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 T_C 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

[50] In this appendix I use the word "branch" to mean something different from the meaning explained in section 2. Here it means 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{
               ⟨KB_new, T_C⟩ ← react to "It's so sunny today" using KB_0;
               aunt_bertha_loop_t(1 hour−T_MJ("It's...ay")−T_C, KB_new)
             }
      else if R = "Looks like rain"
         then{
               ⟨KB_new, T_C⟩ ← react to "Looks like rain" using KB_0;
               aunt_bertha_loop_t(1 hour−T_MJ("Loo...ain")−T_C, KB_new)
             }
      else...
         about 10^445 more branches
      ...
}
```

Fig. 4 Program after branch expansion

```
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.[51] So we can replace the `if` with C, which looks like

```
let R = (read next query from judge ) in
    {
      ⟨KB_new, T_C⟩ ← react to R using KB;
      aunt_bertha_loop_t(T−T_MJ(R)−T_C, 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^{445} will have the variable $R_{10^{445}}$. The result is as shown in figure 6.[52]

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:

```
else if R = string_i
    {
      print response_i;
      R_i ← (read next query from judge);
      if R_i = ## then
         ...
      else if R_i = "It's so sunny today" then
```

[51] Although it's hard to be completely sure of what happens in 10^{445} branches.

[52] How come I haven't had to treat $\mathtt{KB_{new}}$ and $\mathtt{T}_C$ the same way I handled `R`? I could 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, T_C⟩ ← react to R using KB_1;
                   aunt_bertha_loop_t(T_1−T_MJ(R)−T_C, 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, T_C⟩ ← react to R using KB_2;
                   aunt_bertha_loop_t(T_2−T_MJ(R)−T_C, KB_new)
                  }
             }
            }
    else ...
       about 10^445 more branches
    ...
}
```

Fig. 5 Program after further branch expansion

```
    ...
  else ...
  10^445 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^{445} branches; the second, $(10^{445})^2$.

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

```
if R = ...and R_{j_2} = ...and ...R_{j_k} = ...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^{445} branches of that length; then it becomes 2 and there are $(10^{445})^2$ branches of length 2; and so forth up to $k =$ the maximal number of exchanges between judge and examinee that can occur in one hour.

```
{
 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);
              ⟨KB_new, TC⟩ ← react to R1 using KB1;
              aunt_bertha_loop_t(T1−TMJ(R1)−TC, KB_new)
             }
     else if R = "Looks like rain"
        then {
              print "My garden can use it.";
              R2 ← (read next query from judge);
              ⟨KB_new, TC⟩ ← react to R2 using KB2;
              aunt_bertha_loop_t(T2−TMJ(R2)−TC, KB_new)
             }
     else ...
        about 10^445 more branches
     ...
 }
```

Fig. 6 Program after applying `let`-elimination in every branch

```
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
      ...
    ...
   }
```

Fig. 7 Sketch of program after applying `if`-raising to the outer loop.

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.[53] These strings correspond exactly to the key strings used in HTPL.[54] 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, husb)` and `name(husb, "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

[Allen and Kennedy(2001)] Allen R, Kennedy K (2001) Optimizing Compilers for Modern Architectures: A Dependence-based Approach. Morgan Kaufmann, San Francisco

[Bertsekas(1987)] Bertsekas DP (1987) Dynamic Programming, Deterministic and Stochastic Models. Prentice-Hall, Englewood Cliffs, NJ

[Binmore(2007)] Binmore K (2007) Playing for Real: A Text on Game Theory. Oxford University Press

[Block(1978)] Block N (1978) Troubles with functionalism. In: Savage CW (ed) Perception and Cognition: Issues in the Foundation of Psychology, Minn. Studies in the Phil. of Sci, University of Minnesota Press, pp 261–325

[Block(1980)] Block N (ed) (1980) Readings in the Philosophy of Psychology. Harvard University Press, Cambridge, Mass., 2 vols

[Block(1981)] Block N (1981) Psychologism and behaviorism. *The Philosophical Review* 90(1):5–43

[Borges(2000)] Borges JL (2000) The library of Babel. In: The Total Library: Non-Fiction 1922-1986, pp 214–216, translated by Eliot Weinberger

[Botterill and Carruthers(1999)] Botterill G, Carruthers P (1999) The Philosophy of Psychology. Cambridge University Press

[Braddon-Mitchell(2009)] Braddon-Mitchell D (2009) Behavourism. In: Symons J, Calvo P (eds) The Routledge Companion to Philosophy of Psychology, Routledge, London, pp 90–98

[Braddon-Mitchell and Jackson(2007)] Braddon-Mitchell D, Jackson F (2007) Philosophy of Mind and Cognition. Blackwell Publishing, 2nd Edition Oxford

[Braithwaite et al(1952)Braithwaite, Jefferson, Newman, and Turing] Braithwaite R, Jefferson G, Newman M, Turing A (1952) Can automatic machines be said to think? (BBC Radio broadcast.) Also in (Copeland 2004)

[Chisholm(1957)] Chisholm R (1957) Perceiving. Cornell University Press, Ithaca

[Christian(2011)] Christian B (2011) The Most Human Human: What Talking with Computers Teaches Us About What It Means To Be Alive. Doubleday, New York

[Copeland and Proudfoot(2009)] Copeland BJ, Proudfoot D (2009) Turing's test: A philosophical and historical guide. In: [Epstein et al(2008)Epstein, Roberts, and Beber], pp 119–138

[Culbertson(1956)] Culbertson JT (1956) Some uneconomical robots. In: [Shannon and McCarthy(1956)], pp 99–116

[Davidson(1987)] Davidson D (1987) Knowing one's own mind. In: Proc. and Addresses of the Am. Phil. Assoc, vol 60, pp 441–58, (Also in Donald Davidson 2001 *Subjective, Intersubjective, Objective.* New York and Clarendon: Oxford University Press, pp. 15–38.)

[Dennett(1978)] Dennett DC (1978) Toward a cognitive theory of consciousness. In: Dennett DC (ed) Brainstorms, Bradford Books/MIT Press, Cambridge, Mass., pp 149–173, originally in Savage (1978)

[53] 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.

[54] See section 2 for why the length of a key string = the number of judge inputs so far.

[Dennett(1985)] Dennett DC (1985) Can machines think? In: Shafto M (ed) How We Know, Harper & Row, San Francisco, pp 121–145
[Dennett(1995)] Dennett DC (1995) Darwin's Dangerous Idea: Evolution and the Meanings of Life. Simon and Schuster, New York
[Dowe and Hájek(1997)] Dowe DL, Hájek AR (1997) A computational extension to the Turing test. Tech. Rep. 97/322, nil, department of Computer Science, Monash University
[Dowe and Hájek(1998)] Dowe DL, Hájek AR (1998) A non-behavioural, computational extension to the Turing Test. In: Proc. Int. Conf. on Computational Intelligence and Multimedia Applications, pp 101–106, gippsland, Australia
[Epstein et al(2008)Epstein, Roberts, and Beber] Epstein R, Roberts G, Beber G (2008) Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer. Springer
[Fodor(1975)] Fodor J (1975) The Language of Thought. Thomas Y. Crowell, New York
[French(1990)] French RM (1990) Subcognition and the limits of the Turing Test. *Mind* 99(393):53–65, reprinted in (Shieber 2004), pp. 183–197
[Furht and Escalante(2010)] Furht B, Escalante A (eds) (2010) Handbook of Cloud Computing. Springer, New York
[Geach(1957)] Geach P (1957) Mental Acts. Routledge and Kegan Paul, London
[Harnad(1990)] Harnad S (1990) The symbol grounding problem. *Physica D* 42:335–346
[Harnad(1991)] Harnad S (1991) Other bodies, other minds: A machine incarnation of an old philosophical problem. *Minds and Machines* 1(1):43–54
[Harnad(2000)] Harnad S (2000) Minds, machines, and Turing. *J of Logic, Language and Information* 9(4):425–45
[Hayes and Ford(1995)] Hayes P, Ford K (1995) Turing Test considered harmful. In: Proc. Ijcai, vol 14, pp 972–977, vol. 1
[Hodges(1983)] Hodges A (1983) Alan Turing: The Enigma. Simon and Schuster, New York
[Holland(2003)] Holland O (ed) (2003) Machine Consciousness. Imprint Academic, Exeter
[Homer and Selman(2011)] Homer S, Selman AL (2011) Computability and Complexity Theory. Springer, New York
[Humphrys(2008)] Humphrys M (2008) How my program passing the Turing Test. In: [Epstein et al(2008)Epstein, Roberts, and Beber], pp 237–260
[Jones et al(1993)Jones, Gomard, and Sestoft] Jones N, Gomard C, Sestoft P (1993) Partial evaluation and automatic program generation. With chapters by L.O. Andersen and T. Mogensen. Prentice Hall International
[Kam(1997)] Kam T (1997) Synthesis of finite state machines: Functional optimization. Boston: Kluwer Academic Publishers
[Kirk(1995)] Kirk R (1995) How is consciousness possible? In: Metzinger T (ed) Conscious Experience, Ferdinand Schoningh (English edition published by Imprint Academic), pp 391–408
[Knuth(1998)] Knuth DE (1998) The Art of Computer Programming: Seminumerical Algorithms. Addison-Wesley, Reading, MA, (3rd edition)
[Leber(2013)] Leber J (2013) The immortal life of the Enron e-mails. *Technology Review* 116(5):15–16, URL http://www.technologyreview.com/news/515801/the-immortal-life-of-the-enron-e-mails/
[Leigh(2006)] Leigh J (2006) Applied digital control: Theory, design and implementation (second edition). Dover Publications
[Lenat(2009)] Lenat DB (2009) Building a machine smart enough to pass the Turing Test: Could we, should we, will we? In: [Epstein et al(2008)Epstein, Roberts, and Beber], pp 261–282
[Lindholm et al(2012)Lindholm, Yellin, Bracha, and Buckley] Lindholm T, Yellin F, Bracha G, Buckley A (2012) The Java Virtual Machine specification: Java se 7 edition. URL http://docs.oracle.com/javase/specs/jvms/se7/html/index.html, (accessed 2012-07-01)
[McDermott(2001)] McDermott D (2001) Mind and Mechanism. MIT Press, Cambridge, Mass.
[Millican and Clark(1996)] Millican P, Clark A (1996) The legacy of Alan Turing. Oxford: Clarendon Press
[Perlis(2005)] Perlis D (2005) Hawkins on intelligence: fascination and frustration. *Artificial Intelligence* 169:184–191
[Purtill(1971)] Purtill R (1971) Beating the imitation game. *Mind* 80(318):290–94, (Reprinted in Shieber 2004, pp. 165–71.)
[Rothschild(1986)] Rothschild L (1986) The distribution of english dictionary word lengths. *J Of Statistical Planning And Inference* 14(2):311–322
[Russell and Norvig(2010)] Russell S, Norvig P (2010) Artificial Intelligence: A Modern Approach (3rd edition). Prentice Hall
[Searle(1980)] Searle JR (1980) Minds, brains, and program. *The Behavioral and Brain Sciences* 3:417–424
[Shannon(1950a)] Shannon C (1950a) A chess-playing machine. *Scientific American* 182(2):48–51, reprinted in James R. Newman 1956 *The World of Mathematics* **4**, New York: Simon and Schuster, pp. 2124–2133

[Shannon(1950b)] Shannon C (1950b) Programming a computer for playing chess. *Philosophical Magazine* 7-41(314):256–275, reprinted in D. N. L. Levy (ed.) 1988 *Computer Chess Compendium.* New York: Springer-Verlag, NY

[Shannon and McCarthy(1956)] Shannon CE, McCarthy J (eds) (1956) Automata Studies. (Note: *Annals of Mathematics Studies* **34**.) Princeton University Press

[Sloman and Chrisley(2003)] Sloman A, Chrisley R (2003) Virtual machines and consciousness. *J Consciousness Studies* 10(4–5):6–45, reprinted in (Holland 2003), pp. 133–172

[Smith and Di(2009)] Smith S, Di J (2009) Designing asynchronous circuits using NULL conventional logic (ncl). Morgan & Claypool Publishers

[Styler(2011)] Styler W (2011) The EnronSent corpus. Tech. Rep. 01-2011, nil, URL http://verbs.colorado.edu/enronsent/, university of Colorado at Boulder Institute of Cognitive Science

[Turing(1950)] Turing A (1950) Computing machinery and intelligence. *Mind* 49:433–460

[Turing and Brooker(1952)] Turing A, Brooker R (1952) *Programmers' Handbook (2nd Edition) for the Manchester Electronic Computer Mark II.* URL http://www.computer50.org/kgill/mark1/progman.html

[Wegener(1991)] Wegener I (1991) The complexity of boolean functions. Wiley

[Weizenbaum(1976)] Weizenbaum J (1976) Computer Power and Human Reason: From Judgment To Calculation. W. H. Freeman, San Francisco