How Long Can the Angels Keep Dancing in the Chinese Room?

Richard Yee

Version 1.3

January 2016

Contents

"Everyone is entitled to his own opinion, but not to his own facts."
— Daniel Patrick Moynihan

1 Introduction

In 1950, the mathematician Alan Turing famously wrote:

I propose to consider the question, 'Can machines think?'

Then, citing the difficulty of defining the terms 'machine' and 'think', he promptly replaced that question with what he called the 'imitation game'—a.k.a. the *Turing test*—which asks whether a digital computer could be indistinguishable from a human at the level of textual discourse alone [1]. Thirty years later, the philosopher John Searle addressed Turing's questions with his famous *Chinese room argument* (CRA), which claims that even if a computer could pass the Turing test by running some program, that alone could never suffice for the *intentional* semantic understanding of the language input that is necessary for human thinking. If this is right, then

(1) the Turing test is fallible in determining the presence of human thought in computers.

More importantly, Searle claims that because the CRA applies to any digital computer running any program, it shows that instantiating the right program could never by itself be sufficient for equaling human semantic understanding (hence, human thought, consciousness, mind) [2]. In other words, the CRA shows that

(2) the *computational theory of mind* (*computationalism*) is wrong.[1]

Searle dubbed the combination of these two propositions: (1) the Turing test and (2) the computability of mind, as the *strong AI (artificial intelligence)* position [2]. Note however that they are logically independent. Unfortunately, many CRA discussions conflate them, which can further muddy the already murky waters. This discussion only addresses the second proposition, computationalism. It takes no stand on the Turing test.

Since the CRA first appeared, over 35 more years have passed. So is computationalism now dead? Or has the CRA gone the way of angels dancing on pinheads?

Amazingly, both sides are still alive and well. The CRA has always had many critics. They might even represent a consensus, one that generally favors the rebuttal that Searle dubbed the *systems reply* [2] [3]. Despite this, the CRA still enlists many new believers, and it remains perennially on the "greatest hits" charts. How is this possible?

The following discussion refutes the CRA yet again—philosophy's version of a Sisyphean labor. What more can be said? Something mathematical, perhaps? The *theory of computation* makes it obvious that the CRA is fatally flawed—based on an objective, mathematical analysis that does not depend on anyone's subjective opinions or intuitions. Basic knowledge of *Turing machines* clarifies the essential concepts of *computer* and *program*. This unmistakably exposes the CRA's fallacy. It also gives precise accounts of both *formal symbol manipulation* (aka "syntax vs. semantics") and the famous *systems reply* [4] rebuttal. Although the prospect of decisively burying the CRA remains daunting [3], this analysis at least helps to explain the CRA's remarkable endurance.

[1] The only type of computation considered in this discussion is Turing-machine-equivalent.

2 Background: Turing Machines, Programs, and Universal TMs

Assuming prior familiarity with the CRA, this discussion will reference it without recapitulating it. Understanding the CRA's flaw also requires basic knowledge of what *Turing machines* are and how they relate to *universal Turing machines* via *programs*. This section summarizes that background along with pertinent ties back to the CRA. Those who would rather cut to the chase and see about the flaw are free to skip, skim, or return to this section as appropriate.

Turing machines (TMs) are mathematical models of a computation type that is algorithmic, i.e., finitely describable and executable in discrete steps. A universal Turing machine (UTM) is a special subclass of TM, which represents the familiar concept of "programmable computer". Hence, most TMs are not universally programmable. The difference between UTMs (programmable computers) and non-universal TMs (all other computers) is crucial for understanding the CRA.

2.1 Turing Machines

Rather than give mathematical definitions of TMs and UTMs, suffice it here to highlight concepts that are key to understanding the CRA. Figure 1 illustrates a basic TM structure, which applies to all TMs, including UTMs.

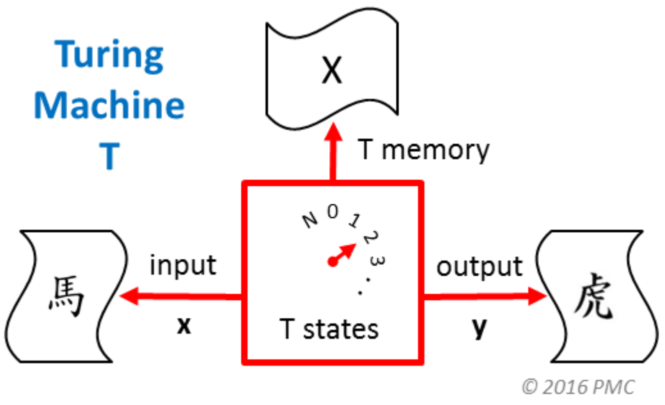

Figure 1: Turing machine *T* has internal states and three tapes. Each transition step updates its state and tapes. A program *Tp* (not shown) can encode *T*'s structure and transition function.

It shows a TM, *T*, whose infinite tape (for reading and writing symbols) is split into three[2] for the following uses:

1. external inputs received by *T*,
2. internal memory/storage used by *T* during its computation steps, and
3. external output produced by *T*.

[2] The standard TM model has just one tape, but multi-tape TMs are equivalent for most purposes, including ours. Moreover, the three tapes serve here solely to clarify the three kinds of data. A standard single-tape TM would work here as well.

It also shows that *T* has a (fixed, finite) set of internal states. Tape contents are strings of symbols from fixed, finite alphabets. T computes via discrete *transition* steps, where at each step:

1. *T* uses its current state and currently positioned tape symbols (one per tape), to cause a single-step change to
2. a new state, new tape symbols, and new tape positions for the next step.

The (finite) set of all transition steps constitutes the TM's *transition function*. Figure 1 does not depict the transition function because it is not a thing; rather it is how the machine operates.

2.2 Programs are TM Blueprints

Because the set of internal states and the tape alphabets are all finite (and processed as single symbols), the set of possible transition steps is also finite. This means that the entire TM, its states, tape symbols, *and* its transition function, can be completely encoded into a finite-length string—better known as a *program*.

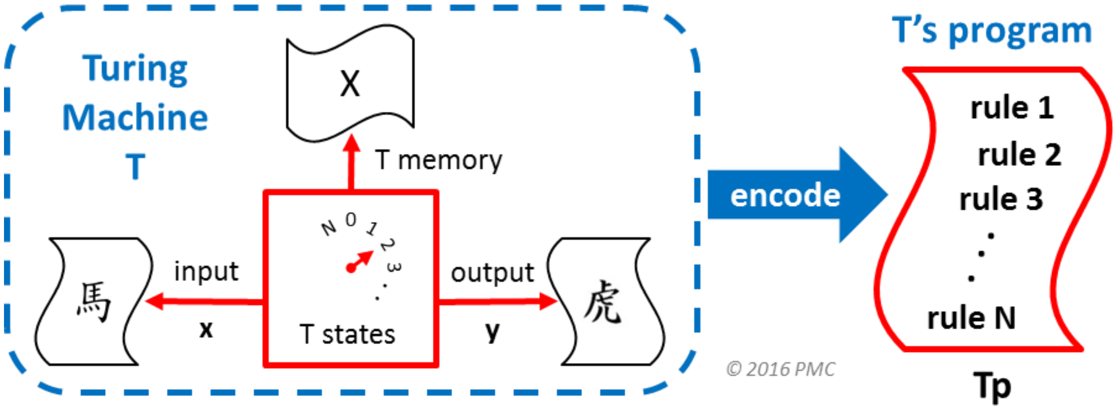

Figure 2: The structure and functioning of Turing machine T is encoded into a program Tp.

Thus, a program is nothing more (or less!) than a complete description of a specific Turing machine. It is like the blueprint of a building. A program Tp describes how machine T: (1) processes its input symbols, (2) maintains its internal memory, (3) updates its machine states, and (4) produces outputs.

Just like a blueprint, for any given a program, one can build a physical device that implements it. Because one can physically implement a program in different ways, TM computations are *multiply realizable*.

2.3 Universal Turing Machines

This brings us to the UTMs, the general contractors of the TM world. UTMs are special because they can (1) interpret any TM's given program and thereby (2) instantiate that TM's computation coincidentally with its own, universal computation. Therefore, instead of building a different physical device for each new TM that comes along, it suffices to build one physical UTM device: a (universally) *programmable computer*. The UTM-computer can instantiate any other TM that one wants to create. One just encodes the desired TM as a program that the UTM-computer can interpret and execute. A UTM can thus reproduce the output of any such programmed TM on any given input.

2.3.1 How do UTMs work?

UTMs *are* TMs, so they are fundamentally the same as described in section 2.1 above. A distinctive and essential feature of a UTM is that its input consists of **two** separate parts: (1) a program and (2) a "nominal (or formal) input". Figure 3 shows that UTM *U* has input *z* consisting of program *Tp* and nominal input *x*, which is what would have been the total input to machine *T* had it been built directly instead of using machine *U* (see *x* in Figure 1).

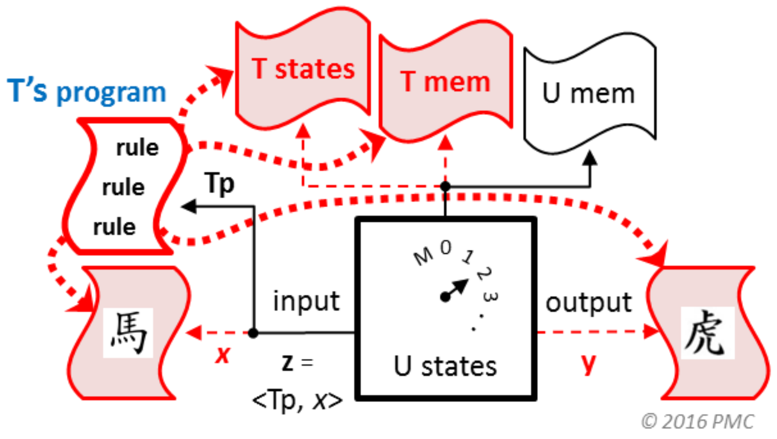

Figure 3: Universal-TM *U* has two-part input *z* with: (1) program *Tp* and (2) nominal (formal) input *x*. *U* interprets *Tp* thereby instantiating *T*'s computation on nominal input x: U(z) = U(<Tp,x>) = *T(x)* = *y*. The red (curved, thick, dotted) arrows from *T*'s program show the transition updates of *T*'s computation (as in Figure 1). This process simultaneously realizes (at least) two distinct TM computations: (1) universal computation U and (2) computation T.

When a UTM runs, it must translate its program's rules into computational actions. The *Tp* rules encode the state and tape transitions of machine *T*, which UTM *U* executes on *T*'s behalf, recording *T*'s states on tape in the process. The red (curved, heavy) arrows in Figure 3 show *T*'s transition updates. They correspond to the red transition arrows shown in Figure 1 for the same TM *T*.

U's universal computation thereby instantiates *T*'s computation—two distinct computations, instantiated simultaneously. *U* computes on <*Tp*, *x*>, producing *T*'s computation on *x* alone, which results in *T*'s output *y*. By holding program *Tp* constant and restricting one's attention to nominal input *x* and output *y*, one observes machine *T*, not only in terms of *T*'s input-output behavior but also in terms of *T*'s computation, which is fully realized.

2.3.2 The slippery term: "Computer"

The term "computer" usually refers to UTMs, but it can also refer to TMs in general. Too often, people use it ambiguously, leaving the reference unclear. UTMs are highly specialized TMs, with some very unusual characteristics. However, because UTMs are so familiar as programmable computers and because they are called "universal", their idiosyncrasies become mistakenly identified with the entire scope of TM computation. This overgeneralization is the primary source of the CRA's mischief.

2.4 Formal Manipulation of Input Symbols

Searle and Harnad make much ado about how the syntactic, formal manipulation of input symbols is insufficient for semantics [2] [5] [6]. Formal manipulation means treating symbols solely as tokens having distinguishable identities based on form/shape and nothing more. The manipulator processes the symbols through shape-matching alone, without further associating them with any of the manipulator's own subjective or intrinsic information, no meaning, no significance to the manipulator beyond their extrinsic forms and their references in externally prescribed rules. Such processing thus entails no *intentionality*[3] on the part of the manipulator. Although an external observer might ascribe meaning to such a process, this would be (1) an external attribution, which could be (2) mistakenly attributing intrinsic semantics based on an illusion of it, mistaking simulated intentionality for the real thing.

2.4.1 UTMs formally manipulate their nominal inputs—only

When the symbol manipulator is a UTM computer, this formal relationship indeed holds between the UTM *U* and its nominal input *x*. A UTM can always treat its nominal inputs as purely formal tokens because it also has an external program, which dictates how to process the *x* symbols properly. Indeed, UTMs *must* treat their nominal inputs this way, to avoid disrupting the program's target computation.

Of course, UTM computers cannot treat their other input, the program, in this same way, i.e., as being entirely composed of purely formal symbols. A computer must actually interpret the program's symbols so that it can cause the proper state and memory transitions to occur.

Consequently, in the CRA, this explains why the rules (Searle's program) had to be in English. One should understand that the CRA's rules in English are part of the total input to the UTM computer, Searle. This fact alone undermines Searle's argument against computers as processing their inputs purely syntactically (formally).

2.4.2 What is—and is *not*—"universal" about UTMs?

The so-called universality of a UTM lies in its ability to partially reproduce the input-output (IO) behavior of any TM, suitably encoded as a program. Such IO reproduction is "partial" because it only pertains to the nominal input, excluding the program input. What is *not* universal about UTMs is the fact that they have such "nominal inputs" which they can and must treat purely formally.

In general, non-universal TMs have no constraints on whether they process their input symbols formally or not. They are free to associate their input symbols with any internal states or data and then proceed accordingly. The sky's the limit for general computers.

This is the crucial concept to grasp regarding the CRA: TM processing of input symbols is not generally required to be purely formal. The fixation on formal symbol processing that surrounds the CRA stems from the exceptional manner in which UTMs process their nominal inputs. The fact that a UTM computation does not associate its nominal inputs to *its own*

[3] *Intentionality* is a special term from philosophy, which pertains to the concepts of reference, representation, or "aboutness".

internal states and memory contents[4] is not a characteristic that is "universal" to all TMs. Indeed, the opposite is generally true.

3 The CRA's Flaw

The theory of computation specifies the precise relationships between a UTM, another TM, and the program of that other TM. It explains how the UTM's execution of the program produces the other TM's computation, which is distinct from its own computation (universal programmability). From that understanding, the flaw in the Chinese room argument is readily apparent.[5]

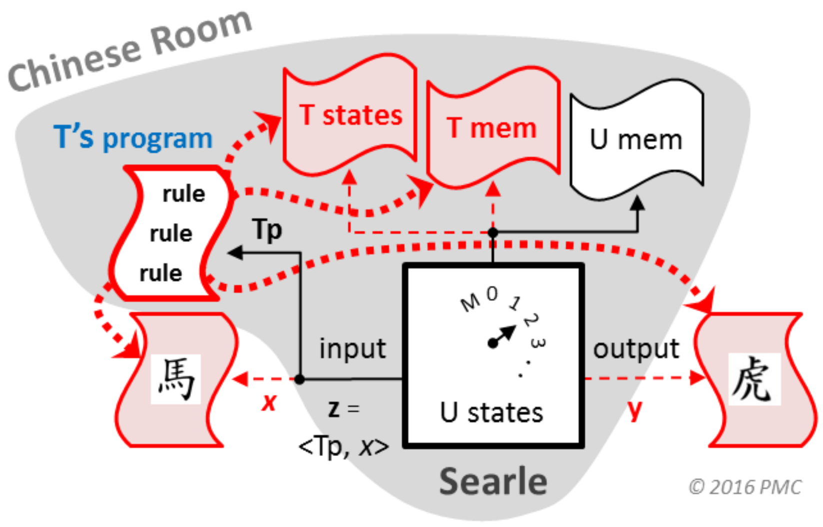

Figure 4: UTM Searle instantiates *T*'s computation. As a UTM, Searle must process the Chinese input symbols formally, but computation *T*'s processing of them is unrestricted.

3.1 In a Nutshell

1. There are two significant and identifiable TM computations occurring in the room:
    a. the computation of a universal Turing machine and
    b. the computation of the TM *T* that is described by the ruleset (program).
2. The person, Searle, implements only the UTM computation. Thus like all UTMs, he processes the two parts of his input in two different ways.
    a. He reads and executes T's program rules.
    b. He formally manipulates the nominal input (Chinese characters) by matching them against the rules and tapes, and then making updates as prescribed.

[4] Of course, acting as an agent on behalf of its program, a UTM does process nominal inputs relative to the program-controlled state and memory data. To the UTM, such processing is formal, whereas to the program's TM it might not be.

[5] This refutation of the CRA is an updated version of one presented in Yee, 1993 [4].

3. TM computation *T* uniquely constitutes the Chinese input-output computation. However, *T*'s algorithm is completely unknown. In particular, the CRA never establishes whether computation *T* processes the Chinese input symbols with or without intentionality.
4. Consequently, the CRA is inconclusive: it neither proves nor disproves the possibility that computation *T* equals a human understanding of the Chinese input.

3.2 Elaboration

- Point #1, regarding the existence of the two computations, does not depend on anyone's opinions or intuitions. It is a mathematical fact about how UTMs work.
- This argument refutes the claim that the CRA disproves computationalism. (Of course, neither does this prove computationalism.)
- This refutation takes no stance on whether the room's internal processing of Chinese equals that of a human. Hence, it argues neither for, nor against the adequacy of the Turing test.
- The key point to understand is #2.
    - Searle, as a UTM-computer, can and must process the Chinese inputs purely formally because they are his nominal inputs (see section 2.4 above).
    - On the other hand, Searle may not manipulate his other inputs, the rules, purely formally because he must interpret and implement them. Thus, he requires these inputs to be in English so that he—*as a computer(!)*—can read and understand the rules in order to execute them properly. In other words, the Searle-computer must process the rule inputs semantically, with intentionality.
        - This is the single biggest irony of Searle's take on computers!
    - Searle's presence is a big red herring.
        - Although Searle's UTM computation might appear to be crucial, it is not at all. For any given program, one could always replace a UTM running it with a machine that implements the program directly (as in Figure 1), thus eliminating the UTM computation. Doing this for the CRA would also remove the UTM-requirement for processing the Chinese inputs purely formally because they would no longer be "nominal" to a UTM.
        - Even with UTM Searle present, his introspection is limited to his own universal computation: the rules, nominal inputs, etc. He does not know whether computation *T* is making any associations between the Chinese symbols and other internal information from the program or from previous input-output interactions. Thus, Searle does not know whether computation *T* is processing the Chinese inputs the way a human Chinese speaker would.
- Point #3 highlights the fact that because the CRA focuses solely on Searle's UTM computation, it completely ignores the only computation that actually matters for the Chinese input-output behavior.

4 The CRA under the TM Spotlight

This TM-based analysis also sheds light on some of the CRA's prominent discussion points.

4.1 What Computations *Can* Searle Detect?

The CRA relies on Searle's consciousness as a kind of detector for an alleged "Chinese understanding" computation. The logic of the argument is this.

1. If computation alone were sufficient for human understanding of Chinese, then in principle there would be a program for it.
2. Let Searle execute such a program exactly as a computer would.
3. In so doing, Searle detects no human understanding of Chinese. Hence, it does not occur in the room.

If Searle could reliably detect a computation for human understanding of Chinese, then he should be able to detect simpler computations as well, right? If his argument works at all, then it should work at least for computations such as integer addition, sorting lists of numbers, or playing tic-tac-toe.

Pick one, *addition* for example, and write a ruleset (program) for it in English. To guard against cheating, encode the inputs and outputs using Chinese characters, which we know will prevent Searle from straying from the rules and applying his personal semantics to them. For example, the Chinese character for "horse" could mean "7" and so forth. Now, with the new ruleset for addition, run the room as before, and then ask Searle, "Are you doing addition?"

Once again, he would say, "No, I'm manipulating meaningless squiggles and squoggles"—and again, he would be correct because that is how the UTM computation feels. He is definitely not doing addition. Then ask him, "Well, if *you* are not doing addition, is addition happening within the room?"

- Would he say, "I don't know"? Maybe, but that is not how he responded in the CRA.
- Would he say, "Yes"? Highly doubtful. On what basis could he establish that? [6]
- Would he say "No" again? If so, we know that he would be wrong.

The point is that regardless of what program he is running, Searle as a UTM cannot reliably detect the presence or absence of any particular computation other than his own: *universal programmability*. Thus, Searle's CRA denial that he could be instantiating a computation for human understanding of Chinese is baseless.

4.2 The Systems Reply: What exactly is "the system"?

The TM analysis upholds part of the popular systems reply by explaining precisely what constitutes "the system" and how it arises. As noted above, this refutation does not assume that such understanding definitely exists. This might be a partial difference with traditional supporters of the systems reply, who might also include the Turing test assertion.

The system that has the *potential* for understanding the Chinese is *T's* computation, which consists of:

- the set of rules (which encodes *T*'s transition function),

[6] If applicable, Rice's Theorem would preclude "yes" responses in general.

- the notes that contain *T*'s states and memory contents, and
- the nominal-input and output messages.

The instantiation of this system constitutes *T*'s computation according to the theory. Note that *T*'s system—the one that might actually understand the Chinese—*does not include Searle!* But again, this is obvious from the theory. Searle is only providing a UTM computation, which for any given program, is always replaceable by a direct realization. The independence of *T*'s computation also explains why if *T* did entail a human's understanding of Chinese, Searle would not become aware of it, as he is just a dispensable "middleman".

Finally, this analysis might also help to explain why the systems reply is so popular. With the program in complete control of all of the relevant information processing, much is happening that Searle cannot fathom from his limited perspective. Many have understood or sensed the significance of the program's scope being beyond the ken of the person alone. To borrow an expression from politics, "It's the program, stupid!"

4.3 Searle's Response to the Systems Reply

Searle's response to the systems reply is to imagine that he memorizes the entire system, so that it operates entirely within his own head. Then there is nothing left but Searle himself, and still he does not understand Chinese.

This maneuver, however, changes nothing of significance. All it does is relocate the program and the tapes without changing Searle's disposition toward them. Searle remains a UTM. The program in his memory remains a program to him, and rote recall of the program rules is no different than reading them. Thus, when Searle again executes his UTM computation, T's computation is again instantiated, albeit in a new, squishier medium.

Regarding the systems reply, Searle writes:

> *Actually I feel somewhat embarrassed to give even this answer to the systems theory because the theory seems to me so implausible to start with. The idea is that while a person doesn't understand Chinese, somehow the conjunction of that person and bits of paper might understand Chinese. It is not easy for me to imagine how someone who was not in the grip of an ideology would find the idea at all plausible. [2]*

Guilty as charged perhaps—although what is gripping is not so much an ideology as it is a mathematical theory. The basis of the systems reply is not some vague notion requiring a leap of faith. It is objectively explainable from the theory, and it only requires understanding.

5 Searle's Siren Song: Computer as UTM

Moving beyond the details of CRA debates, Searle constructs a summary of his refutation of strong AI's two propositions: the Turing test and computationalism. The foundation of this refutation is the CRA's demonstration that formal symbol manipulation lacks intentional semantics. This foundation, however, pertains only to UTM computers, which helps to explain why the CRA has been both so compelling and so perplexing.

5.1 Overgeneralizing the Formal Symbol Processing of UTMs

Searle's builds his refutation of computationalism on three premises.

> *Premise 1: Implemented programs are syntactical processes.*
>
> *Premise 2: Minds have semantic contents.*

> *Premise 3: Syntax by itself is neither sufficient for nor constitutive of semantics.*

These produce his anti-computationalist result.

> *Conclusion: Therefore, the implemented programs are not by themselves constitutive of, nor sufficient for, minds. In short, Strong Artificial Intelligence is false. [5]*

How does the flaw in the CRA undermine this argument and preserve the possibility of computationalism? In Premise 1, it is unclear what kind of program implementation Searle intends. If he means a UTM running a program, then the syntactical processing of a UTM on its nominal inputs was already addressed in section 2.4 above. In this case, Premise 1 pertains to UTM computations only, rather than to TMs in general, and computationalism remains unscathed.

However, if Searle means for Premise 1 to pertain to all TM computation, then from his discussion of it he means simply that programs are logical TM specifications, and they are multiply realizable, just as described in section 2.2 above. So then, the focus shifts to Premise 3. To justify Premise 3 Searle writes:

> *The Chinese Room thought experiment illustrates this truth. The purely syntactical operations of the computer program are not by themselves sufficient either to constitute, nor to guarantee the presence of, semantic content, of the sort that is associated with human understanding. The purpose of the Chinese Room thought experiment was to dramatically illustrate this point. It is obvious in the thought experiment that the man has all the syntax necessary to answer questions in Chinese, but he still does not understand a word of Chinese. [5]*

Section 3 refuted this by explaining how the CRA fails to prove the absence of a human level of semantic understanding of Chinese. The central problem of the CRA stems from Searle's failure to distinguish program *T*'s TM computation from his own UTM computation. In particular, the only Chinese symbol processing that Searle recognizes is his own formal UTM processing, but that has absolutely no bearing on the manner in which computation *T* processes the Chinese inputs. It is free to process them non-formally. Consequently, Premise 3 remains unsubstantiated.

In the CRA, Searle recounts how the thought experiment emerged from his reaction to some language processing research done within the *symbolic AI* paradigm, which at that time had been the dominant paradigm for at least a decade. The generality of the CRA formulation, however, made it applicable to any UTM computer running any program. In its original context, the CRA might have been valuable in highlighting reasonable concerns about over-interpreting the significance of word-level symbol processing on the part of symbolic AI programs. The CRA might even have been reasonable for discussing the special nature of formal symbol processing by UTMs.

However, big problems arose when the CRA's claims were overgeneralized to attack all possible types of TM computation. Usually, such overgeneralizations were made implicitly. (After all, UTMs *are* universal, right? cf. [4]) Unfortunately, these overgeneralizations have been entangling the unsuspecting for well over a generation.

5.2 Simulation vs. Duplication: Computation and Intentionality

Searle's CRA summary also emphasizes that *simulation is not duplication* [5]. To ascertain whether a system has human-level understanding, observing only its input-output behavior is insufficient because intentional semantics is an internal, subjective phenomenon. One can be fooled by external behaviors and mistakenly ascribe internal semantics to a system that has none. Hence, the Turing test is inadequate for identifying it.

If taken only this far, this position is not a current concern because it leaves open the possibility that some type of computation could have valid semantics (albeit requiring some yet unspecified means to verify it). However, Searle further believes the CRA proves that *no* computation could ever have sufficient semantics. He points out that computer-simulated fires, rainstorms, and digestion, do not cause any actual burning, wetness, or indigestion. Thus, as with fire, rain, etc., computations similarly lack “the right stuff” for intentionality. The best one could do, according to Searle, is to simulate intelligence in the sense of *weak AI* [2], creating only the illusion of it.

Of course, the CRA is actually inconclusive, so the computability of intentionality remains an open question. Furthermore, unlike the examples cited in Searle’s simulation analogies, computational processes are logical, not physical.[7] When one computation “simulates” another at the algorithmic (transition function) level, and not just at the input-output level, then that is an actual instantiation (a realization), not merely a “simulation”. A UTM computation instantiates—truly *duplicates*—the computations of other TMs. So if minds *are* computable, then executing their programs would in fact instantiate (realize) them, and not merely simulate them functionally (at the input-output level).

5.3 The Computer Stereotype

Why is the CRA so seductive? How has it captivated novice and expert alike for such an astounding length of time? Fascination over the possibility of intelligent machines has a long history. In a world now driven by technology, the prospect of sentient computers has a special resonance and a prominent place in popular culture. Everyone is familiar with computers and programs, the concepts of hardware and software. On the other hand, the theory of general Turing machine computation is relatively obscure, even to specialized practitioners such as technology professionals and scholars. This imbalance in perspectives fosters a pervasive stereotype that says, “Computers are UTMs”. Table 1 lists some of the key distortions of this view that are most pertinent for discussions of computationalism.

Characteristic	UTM	Non-U TM	Comments
Always governed by an external program	True	False	Most TMs are not programmable, especially not universally.
Required to process its nominal inputs purely as formal tokens	True	False	Most TMs have no “nominal (formal) inputs” (ones intended for a different TM), and they may process inputs as they please.
Required to always process a given input in exactly the same way every time.	True	False	UTMs can never alter their universal programmability function.
Learning & Control: Can use I/O	False	True	Some TMs can interactively adapt their

[7] Physical systems can of course approximately realize computations in the same sense that buildings can approximately realize theories of geometry, statics, etc.

experiences to change its behaviors and interactions with external systems		🟩	internal configurations, which can modify their output behaviors and thus affect their future I/O experiences.

Table 1: Key differences between UTMs and non-universal TMs affect opinions about "computers".

For many, the UTM stereotype engenders a view of computers as dumb machines that are simply slaves to the programs we give them. They can do nothing more than mechanically churn through their programs and inputs like mindless automatons. Nothing about the external world has—nor could have—any intrinsic meaning to them. The CRA is the world's greatest advertisement for this computer-as-UTM stereotype. And Searle is one of its greatest evangelists.

Others see the power of programs. Recall that programs are "TM blueprints", which represent *all* possible classes of TM computers. However, even proponents of computationalism might not always recognize programs in this way. Thus, even would-be TM sympathizers sometimes needlessly surrender key parts of their arguments to the tyranny of the UTM stereotype.

With these powerful forces in all in play:

1. the ubiquity of computers and programs,
2. the compelling (and daunting) prospect of machine intelligence,
3. the pervasiveness of the UTM stereotype,
4. the obvious power of programming, and
5. the obscurity and subtlety of Turing machine theory,

the CRA has long been perfectly positioned to pump everyone's "computer" intuitions (cf. [7]), enticing legions of unsuspecting enthusiasts to dash themselves upon the rocks of fruitless debate.

6 Avoiding the Rocks: Computer as TM

Searle's view of computers as mere syntax machines is widely shared because UTM-computers are the shiniest objects in the world of information technology. However, the CRA's single-minded focus on the Searle UTM computer[8] is the argument's downfall because it ignores the TM computation that is instantiated by executing the program. To some, this might feel like a vindication—the systems reply has been right all along. To others, it might seem like a mere technicality, but the spirit of the argument remains valid: computers cannot think because [*insert favorite intuition here*]. In either case, problems arise when people allow their view of the highly specialized and unrepresentative subclass of UTM computation to become their generalized view of all potential types of TM computation—a mistake no doubt exacerbated by the perilous moniker "universal". It is little wonder then that differing intuitions about computers have fueled no end of passionate, often misplaced, philosophical debate.

What then is a more accurate view of what computers are and what capabilities they might have? For starters, computers as UTMs probably have little significance for most

[8] Pun intended.

philosophical issues because they are dispensable conveniences that have a singular nature. Also, be very leery of “brain as computer / hardware” analogies.

Second, a program (software) is not very interesting either as an artifact or in its relation to a UTM computer—just as the blueprint of a building is not very interesting either as a picture or as a specification to a contractor. In both cases, the resulting physical realization matters much more than the means of producing it. And if nothing else, it should be clear that a UTM computer running a program actually instantiates two computations (1) its own, for universal programmability, and (2) the computation of the TM described by the program.

Beyond that, the outlook for computers as TMs is about as diverse as anything one can cook up. To narrow the scope somewhat for the purposes of philosophy of mind discussions, a good starting point is a topic that figured prominently in Turing’s paper: learning machines [1]. Unlike UTMs, which must repeatedly function the same way every time, some TMs can learn over time from their input-output experiences. And they can change both their inner states and external responses as they do so. This non-UTM ability is obviously fundamental to intelligence. Another closely related perspective on intelligent TMs comes from robotics, where TM computations underlie the processing of myriad sensory inputs and control outputs. Recognizing objects and sounds, planning movements, achieving goals, are all aspects of intelligence that have a computational basis. Far from being governed by externally specified programs, intelligent TMs depend only on their own internal configurations, which they can change through their interactions with the world. Just like us.

7 Acknowledgements

I thank Professor William G. Lycan[9] for reviewing an earlier version of this paper and for many helpful discussions of this and related topics.

8 Bibliography

[1] A. M. Turing, "Computing Machinery and Intelligence," *Mind,* vol. LIX, no. 236, p. 433–460, October 1950.

[2] J. R. Searle, "Minds, Brains, and Programs," *Behavioral and Brain Sciences,* vol. 3, no. 03, pp. 417-424, September 1980.

[3] R. I. Damper, "The Chinese Room Argument: Dead but not yet Buried," *Journal of Consciousness Studies,* vol. 11, no. 5-6, pp. 159-169, 2004.

[4] R. Yee, "Turing Machines And Semantic Symbol Processing: Why Real Computers Don't Mind Chinese Emperors," *Lyceum,* vol. 5, no. 1, pp. 37-59, Spring 1993.

[5] J. R. Searle, "Chinese room argument," *Scholarpedia,* vol. 4, no. 8, p. 3100, 2009.

[6] S. Harnad, "The Symbol Grounding Problem," *Physica D,* vol. 42, pp. 335-346, 1990.

[7] D. C. Dennett, Consciousness Explained, Little, Brown and Co., 1991.

[9] Professor Lycan is the William Rand Kenan, Jr. Distinguished Professor, Department of Philosophy, University of North Carolina at Chapel Hill.