The Social Furniture of Virtual Worlds

Peter Ludlow
University of Campinas (UNICAMP)

DOI: 10.2478/disp-2019-0009 BIBLID [0873-626X (2019) 55; pp.345–369]

Abstract
David Chalmers argues that virtual objects exist in the form of data structures that have causal powers. I argue that there is a large class of virtual objects that are social objects and that do not depend upon data structures for their existence. I also argue that data structures are themselves fundamentally social objects. Thus, virtual objects are fundamentally social objects.

Keywords
Virtual worlds, computation, data structures, social objects, virtual objects.

A common view about virtual worlds (or "synthetic worlds" as Ted Castronova (2005) prefers to call them) is that what transpires in them is either "not real" or "fake" or a "mere simulation" of things that take place on our "real world." And to be sure, if a Second Life friend invites you over to an in-world sushi dinner, there will be no rice or fish entering your physical body. And the virtual sushi may be presented on a virtual table, but you needn't worry about bruising your shin on the table.

On the other hand, while it might not be real edible sushi and a real shin-bruising wooden table, it might be something real for all that. Maybe there is a different way in which these things are real. They aren't real sushi on a real table but they are a real something else.

This is what David Chalmers advocates in his (2017) paper "The Virtual and the Real." On his view, the virtual table and sushi are real "data structures." They are "digital objects" existing on the Second Life servers and the users' client software. The table can't bruise your shin but it has other causal powers. It can disrupt the movement of your avatar, for example, and it can serve as a resting place for your virtual sushi in virtual space. It can determine what is rendered for you on your graphical user interface. The data structure affects the computations performed by the Second Life servers and on your desktop client. It is thus definitely in the mix of real world happenings.

In a little bit, I'm going to ask whether "data structures" is the right way to think about these digital objects, so let's find a neutral way of talking that captures the idea that digital objects correspond to semi-stable states of a computational system. Let's call them "computational structures" for now.

It is worth noting that the computational structures of digital objects are important. The computational structures in robots determine what they will do, and the computational structures in the onboard computer in your car control things ranging from your mileage to warnings about nearby objects. If computers have causal powers (and who can deny that they do) then surely computational structures must have causal powers as well.

Chalmers' story concerns virtual physical objects like tables and plates of sushi, but there are other kinds of objects one might consider as well—for example, social objects like clubs and concerts and corporations. Second Life has these too, but the funny thing is that they don't need computational structures to perpetuate their existence. A Second Life corporation or club can continue to exist long after Second Life shuts down its last server.

I'm going to ask whether computational structures are all that ontologically stable. I'll argue that, contrary to commonly held assumptions, there is no independent fact of the matter grounding the data structures (and computational structures) that a computer has. What existence data structures have is parasitic upon the objects they represent, whether those objects are real world physical objects or virtual world social objects. The virtual objects of Second Life, I contend, are more fundamental than the computational structures on the Second Life servers and our desktop clients. This line of argumentation is going to take us into the teeth of a thesis about the nature of computation, offered by Chalmers (1996) that we shall have to address.

My thought is that social objects provide a deeper insight into virtual world ontology than the virtual physical objects that interest Chalmers. Indeed, I am going to suggest that virtual "physical" objects like tables and plates of sushi just are social objects. They are part of the social furniture of virtual worlds.

1 Social objects in virtual worlds

One of the first things I encountered in The Sims Online, back in 2003, was that users had created a number of in-world institutions, practices, and social artifacts (construed broadly enough to include contracts, and of course currency).

Here is an example of what I mean by an in-world social institution. A number of users in the Sims Online created criminal gangs, modeled on Mafia organizations for the most part, and used those gangs as instruments for the griefing (and in some cases extortion) of other players in the game. These organizations came complete with organizational charts, featuring capos and captains and various other forms of gangland titles. Some groups emerged as responses to those virtual criminal gangs, including one that called itself The Sims Shadow Government, or SSG.

Like most virtual gangs, the SSG had a robust organizational structure and it also had a robust communications network, and not a few spies scattered throughout the Alphaville shard of The Sims Online.

The SSG was clearly a virtual organization. It didn't have physical offices or real guns and they never slapped someone in their physical face, but I don't see that they weren't for all that quite real and quite capable of making your day better or worse than it started.

The SSG was interesting in a lot of ways, but one of the ways in which it was most interesting is that it was portable. That is, the organization was capable of moving to another platform entirely, and as the members grew tired of The Sims Online, they moved their operations to a multiuser game called Star Wars Galaxies.

As far as I know it never happened, but it would have been entirely possible for members of the SSG to begin meeting in physical space and take up physical arms and engage in similar vigilante-style operations in the so-called real life (Let's call it RL). What all this means is that whatever kind of object the SSG is, it is portable between worlds, and it can even exist in RL.

Virtual world currency is another social object that is portable. As early as The Sims Online there were currency exchanges that would offer rates of exchange between the currency of The Sims Online—Simoleans—and US Dollars or other game currencies. This practice continues in virtual worlds like Second Life, where there are currency exchanges that list the daily rate of exchange. But there are also markets in which you can buy non-virtual goods with your game currency (in Second Life, called Linden Dollars). So, for example, you can buy tee shirts and coffee mugs with Linden Dollars. Linden dollars have all the features that money is supposed to have: They are a store of value, a measure of value, and a medium of exchange.

How did that happen? How did something that began, supposedly, as a work of fiction in a virtual world become something that clearly has economic value and can be used to purchase physical items in online markets? The answer, of course, is that the currency has value because of the virtual goods it can be used to purchase, and that value is not restricted to the realm of the virtual. Value is value, and the price point of virtual goods is established by their value relative to other, nonvirtual, goods.

The inventory of portable social objects is not limited to money and vigilante organizations. There are virtual corporations and contracts and works of art and, as Chalmers (2017) notes, there are also mixed media events—concerts for example, which might take place partly in Second Life and partly at a concert stage in RW. Where is the concert? Well, the best first guess would be that it is in both places.

This leads us to consider the trans-world existence of social objects. They really are not dependent on a particular material substrate for the most part. Groups and corporations and currencies can exist in New York City just as easily as they can in Second life.

The first lesson of this should be that whatever *social* objects are, and whatever social properties are, they are not things that depend on their data structures for their existence and they cannot be identified with data structures (or, to use my more neutral way of talking) computational structures. To be sure computational structures can play a role in the way we confront those objects when we are in a virtual world. But I am inclined to say that they are serving to *represent* those virtual objects, or at a minimum they assist in the rendering of representations.

I think that Chalmers might agree with much of what I have said so far in this section—as long as it is restricted to social objects.

That is, I think that he thinks social objects are not identified with data structures in the same way that virtual chairs and virtual sushi are. However, this is the point where we are going to part company, for I believe that understanding social objects in virtual worlds is key to our understanding (virtual) physical objects in virtual worlds. What I aim to suggest is that those objects too—the virtual tables and virtual sushi—have an existence that is quite independent of data structures/computational structures. The fact that they are not portable should not concern us.

To get to this conclusion we are first going to have to do some heavy lifting on the issue of data structures and computational structures, for I am concerned that these things, whatever they are, cannot do the job that Chalmers wishes they could do. Before we get to that, however, I should say some things about virtual physical objects like tables and plates of sushi, because those objects are weird.

2 Virtual "physical" objects

So, what can we say about virtual tables and virtual sushi? To make this discussion productive, let's take a very simple example of a virtual physical object. In Second Life, for example, you can create a simple object or "prim" (as in "primitive object") by rezzing it in-world. There are certain ways you can do this, but one way is simply to point to a certain location in-world and use a pull-down menu to create your object. When you do this, the menu will offer options for your primitive object; you will have the option of choosing the object's primitive shape, for example, and you can chose the "material" constitution of the object (rubber or stone, for example). Depending on the material you choose, the object will behave differently within the virtual physics of the world. If you make your object from rubber, for example, it will bounce around more if you drop it. And that is the other thing to consider. You don't necessarily have to set the object to "physical". You can just rez it in space and it won't fall to the ground until you set it to physical. You might also choose to make your object transparent or you might even make it physically transparent so that people can walk through it.

In Second Life each created object receives a unique identifying number, and herein lies one oddity about virtual objects; they have virtually no properties essentially. You can create an object that is square and has the properties of wood and is set to physical so that it falls to the virtual terra, but you can edit that object so that it is made of glass, is non-physical so that if floats in space, and is a ball or an egg shape. If we want to talk about the causal powers of virtual objects then, we need to talk about their properties at a particular time, or, if you prefer, under a particular edit.

Unlike the causal powers of RL objects, in worlds like Second Life those properties appear to be a function of numerical values that are assigned to the object at a given time. So, for example, the reason the rubber block bounces higher than the stone block has nothing to do with material elasticity and everything to do with numerical values of bounciness that are assigned to the different blocks.

Some properties of the object, like the sound they make when they bounce (glass versus rubber, for example) are a function of scripts that are inserted into the objects to make the relevant sounds. So, for example, if you wanted to you could insert a script into a rubber block that would cause it to make a clinking sound like glass when it bounced.

Most of what these objects do is a function of the scripts that are placed into them. So, for example, a spinning ball will have a spinning script associated with it. The complexity of the actions that the objects can engage in thus are only limited by the Second Life scripting language (LSL), which is fairly robust (it is based on C and Java) and can encode certain basic AI behaviors. In this way, at least, it makes sense to think of the virtual objects as having causal powers in the same sense that computer programs do. On the other hand, one should keep in mind that the script isn't actually *in* the object—it is just called by the object, so there is a question here as to whether whatever the scripting language does counts as something the object is doing or as something that the object is being made to do by a separate program identified with the object.

If you want to identify the scripts for the object as among its causal powers, then things can get weird in other ways. For example, many virtual world objects control the behavior of avatars (as opposed to RL where *we* tend to control the objects). So, for example, in RL, if you go to the stove to cook something, you choose to take out a pan and put it on the stove and put food in the pan. The stove isn't telling you what to do. In the Sims (in all versions—not just the Online version), the stove actually tells your avatar what to do. That is to say, rather than associate cooking and other behaviors with the avatar, the Sims people (Maxis/Electronic Arts) associated that information with the object and the object takes control of the avatar and instructs it go through its cooking and other motions.

The same thing can be true in Second Life. Objects can animate avatars. I suppose the most notorious examples of this would be the sex beds that Stroker Serpentine sold in Second Life—virtual beds that would put two avatars through simulated sex acts, sometimes offering a pulldown menu with options drawn from the Kama Sutra.

I draw on this example to illustrate the kinds of questions that might arise once we start thinking about the causal powers of virtual objects. But of course, Chalmers (2017) did not say that the virtual objects themselves had causal powers, he said that data structures did, but this raises all sorts of questions about what data structures are in general and what they might be in the case of virtual objects.

3 The trouble with data structures

In the introduction, I suggested that we back away from talking about data structures and that we talk about computational structures instead. There are several reasons for this. The first concern I have is that philosophers and computer scientists use 'data structures' in different ways. With an author like Chalmers either use could be in play.

For a computer scientist, a data structure is a way in which data – or more accurately data values are stored, and how those values are related by the computer program. So, for example, a data structure might encode the numerical "name" of the object and the values of the different features objects can have in Second Life. For example, I might create an object in Second Life and assign it certain properties. The data structure encodes the object number, the properties the object has at a given moment and some "hash function" or opaque algorithm connecting the two. Typically, for a platform like Second Life, the data structures are at least partially located on "asset servers" and this raises the question as to whether each asset server organizes the data in the same way. Data structures can be organized in different ways, including arrays, lists, etc.

We can think of virtual world objects in terms of data structures in this way, but as we will see it is a little bit weird to think of these data structures having causal powers—or at least the right kinds of causal powers. Let's begin by thinking about objects in Second Life. If you create a virtual object in Second Life you might begin by "rezzing" a cube. You can then change the values of the object, including its physical properties (rubber or concrete), its dimensions, its surface texture, and whether it is going to be a physical object or not (that is, whether it will obey gravity and whether it will serve as an obstacle to other avatars). If you save that object, it will have a unique object number and it will be stored on a Second Life asset server. It is entirely natural to think of this information (and the computational process linking the object number and its properties) as constituting a data structure in Second Life.

Here is the first thing that is weird about Second Life objects, understood as data structures. There is nothing in that data structure itself that tells us how the object is going to be seen to other players. How the object is seen depends upon the "viewer" or "client software" that is being used. That viewer or software will interpret the data structure and render an image to the user.

Here is an example of how client software can change things up. In the Sims Online, there was a stock set of avatar shapes and a stock set of clothing. But someone wrote a mod to the client-side software, which rendered very different looking avatars. For example, a turtle neck sweater might be rendered as a goth-looking leather coat (there is a picture of my avatar in such a coat on page 6 of Ludlow and Wallace 2007). Of course, client-side software might be modded in any number of ways, so that it is entirely unpredictable how the server side data structures will be interpreted. Some mods in The Sims Online rendered naked avatars, for example. The data structures only provide attributes with numerical values after all.

This is why I think there is something weird about assigning causal powers to data structures, construed in the normal understanding of data structures and causal powers. The data structures just encode information, and how that information is acted on or rendered depends on any number of factors, including the most unpredictable of all—how humans choose to modify their client software.

You might want to say that data structures, construed in this way, only have defined causal powers in the context of a complete computational system, which would mean that the causal powers of the data structure would depend upon some fusion of server-side and client-side software, and of course this would entail that a single object (identified with a unique object number) would have different causal powers for different users at different times. Or perhaps even for the same user. I used to keep two computers running The Sims Online, one with modded software and one without, so that the same avatar would be rendered in different ways on my two computers.

A related issue applies to the server-side data structures. As I noted above, virtual worlds can have multiple asset servers, so that the relevant data structure is distributed—or at least stored in multiple places. But here is the thing: It is easy to imagine a case where different asset servers use completely different kinds of data structures (for example an array on one server and a linked list on another) and we rely on the viewer (the client-side software) to be able to interpret both kinds of data structures. Now, when you layer the possibility of client-side modding on top of that, we can envision a case in which the same object could have completely different data structures and be rendered in completely different ways at the same time and the same virtual location.

I'm not saying one can't make sense of this, but I think we now need a better understanding of what causal powers in general are, and certainly about what causal powers are for virtual world data structures. I certainly feel some pull to say that they don't have any obvious causal powers—they are just lists or arrays of numerical values. Something obviously has causal powers but data structures aren't the obvious candidates, especially when we start thinking in terms of counterfactual dependencies and so forth.

There is another worry about data structures in this sense, which is that they are often eliminable. For example, an optimizing compiler can take a high-level program (like C) and its data structures and collapse them into a more efficient program that might dispense with data structures altogether, or at least eliminate some of the data structures and fold their information into the algorithm. Applied to a virtual world, the idea would be that after objects are created the server and client software could be compiled in a way that dispensed with the data structures. It would still deliver the same "objects" to the screen however. And for our (philosophical) purposes we don't even need an optimizing compiler. To make our philosophical point we only need a compiler that replaced the data structures with a smaller set of data structures and a complex algorithm). It wouldn't matter to the philosophical point that the resulting algorithm was less efficient.

Now, it is entirely possible that Chalmers meant something different by data structure—philosophers (myself certainly included) often talk of data structures as being semi-stable local syntactic states of a computational system. This is a pretty squishy idea, and it pretty much needs to be squishy to work for objects in virtual worlds. The notion of a "local state" would have to include a distributed state that is inclusive of both the servers and the client-side computer. "Semi-stable states" would have to be understood in a way that those states survived the client software being turned on and off quite frequently. I don't think this is impossible to do—the definition just needs to be cleaned up. Just to distinguish this idea from the traditional understanding of data structures, I'm going to speak in terms of *computational structures*. The key difference from the computer scientist's notion of data structure is that a computational structure would include much more information about the architecture and states of the processing system (understood to be a distributed processing system that includes states of both server and client).

Could we understand virtual objects as computational structures in this way, and would it make sense to say that such computational structures have causal powers? Well, yes, but it would seem to entail that we get a kind of fission of virtual objects. If the computational structure supervenes on states of the server *and* the client, then we seem to have different virtual objects for each mod. Maybe the virtual object needs to be understood disjunctively? Or maybe there are lots of computational objects and they don't line up with our informal talk of a single table or a single plate of sushi.

Once everything is cleaned up, however, it all still rests on an assumption—the assumption that there is a fact of the matter about the computational states of the system in isolation. By that I mean that Chalmers' idea of digital objects rests on the idea that the states of the computer, once causally fixed, are what they are quite independently of whatever is going on outside of the computer (or server-client system, in our case). Here I am less sure that we have a fundamentally coherent idea.

I tend towards being an externalist about computation—not in the sense of Clark and Chalmers 1998; my idea is independent of whether notebooks and smartphones are part of our minds. The idea is rather that computational states depend upon environmental embedding. On that line of thinking, something is a syntactic state of a computer, or a computational structure, or the computational state of the fusion of you and your iPhone, only by virtue of such states representing something external to that system. The system-external content is metaphysically prior. Computers can still represent things that don't exist, but they can't represent things that don't bottom out in properties and things that do exist.

Note that we aren't talking about vanilla externalism about content here. I'm not merely saying that what a computational state *represents* depends upon environmental embedding. I mean that whether something is a syntactic state (or computational state) depends upon environmental embedding.

Now, what I'm saying is controversial, because it dances around the idea that there are no computational states in isolation, or as some people have unhelpfully put it: Any physical system can instantiate any finite state computer program. Chalmers (1996) has addressed this idea in the past, but sometimes I wonder if the advocates of the view have put forward the best case for it. The most infamous versions are no doubt Searle (1980, 1990), and Putnam (1988), but I think that the stronger case comes from Kripke (1982) in his reconstruction of Wittgenstein's (1991) rule following argument. I'll get to those arguments in a bit.

First, why would anyone think that any physical system can instantiate any finite state automaton? I think the basic idea is simple enough. A simple block of wood lying on a desk might be used to perform computations. Suppose that the input consists of my pushing the block with my finger and the output consists of the final resting place of the block on my desk. Then the syntactic states of the machine (here taken as the system including the table and the block) supervene on those inputs and outputs and the known surface friction of known irregularities in the block and the table. Hypothetically, there could be creatures for whom making the appropriate finger movements and "reading" the final resting positions of the rock on the table would be simple (imagine someone that memorized the markings on a slide rule—and their positions—and could "read" the inputs and outputs of the blank slide rule). Perhaps less persuasively, Searle has argued that even a wall can instantiate a program like the "Wordstar" word processing program, which I'm not entirely sure is still in use, so think of Word.

Chalmers (1996) argues that Searle goes too far here. According to Chalmers, what we should be interested in here is whether a physical system instantiates a *combinatorial state automaton*, or CSA. Such automata differ from finite state automata in that the internal states of a CSA have combinatorial structure. In particular, as Chalmers puts it, its internal state is a *vector* [S^1, S^2, ..., S^n], where "the *i*th component of the vector can take on a finite number of different values, or *substates*. ... The substates correspond to symbols in those squares or particular values for the cells. …. State-transition rules are determined by specifying for each component of the state vector a function by which its new substrate depends on the old overall state vector and input vector (it will frequently depend on only a few "nearby" components of these vectors), and the same for each element of the output vector." (2017: 324).

So far this sounds right. The processes we are interested in in cognitive science involve computational systems that have variable (but discrete) substates. Those substates in turn, depending on their values, can impact the computations in important ways. This then leads to the question of what it would mean for a physical system to *implement* a CSA, understood as above. Chalmers offers the following proposal.

> A physical system implements a given CSA if there is a decomposition of its internal states into substates [s^1, s^2, ..., s^n], and a mapping *f* from these substates onto corresponding substates S^j of the CSA, along with similar mappings for inputs and outputs, such that: for every formal state transition ([I^1,...,I^k],[S^1,...,S^n])→([S'^1,...,S'^n],[O^1,...,O^l]) of the CSA, if the system is in internal state [s^1,..., s^n] and receiving input [i^1,..., i^n] such that the physical states and inputs map to the formal states and inputs, this causes it to enter an internal state and produce an output that map appropriately to the required formal state and output. (Chalmers 2017: 325)

If we think about the implementation conditions of CSAs in this way then Putnam-and-Searle-type arguments are certainly more difficult to get up and running. For example, the physical system would not only have to mirror the structure of the CSA in a given state, but the sub-states of the physical system would have to be able to encode different values and the substates would have to be connected in the right way. For example, in the example above where we pushed a rock across the table, it is not enough that there be states of the table that are isomorphic to those of the CSM and that the system pass through those states in the right order. Those states would have to be capable of taking different values and they would have to be causally connected in the right way. I assume that this also means that the dependencies between the states would have to be non-accidental. There have to be laws connecting the state transitions of our rock/table computer.

Given these constraints it seems implausible to think that *any* physical object could instantiate any computer program, and it is really hard to see how a wall might implement a computer program like Word. It may even be correct to suggest, as Chalmers does, that "the right sort of complex structure will be found in very few physical systems." I don't know if that's right or not, but we can set that question aside. The real issue is not how many physical systems can instantiate a program, but rather, given a system that is sufficiently complex to implement *one* CSA, whether the system in isolation is enough to determine *which* CSA is being instantiated.

Consider the question of whether a given physical computer is executing a procedural or a declarative language. Here, we are assuming that we don't have access to the intentions of the programmer, and no way to download whatever high-level program is being executed (or no reason to trust the contents of such a download).

As a canonical example of a declarative language we might take Prolog, and as a canonical example of a procedural language we might take C or Pascal.

Biermann (1997) describes the difference this way:

Most programs are of the form

Do this.
Do that.

> Do something else.
> Etc.
>
> Thus the programmer uses the program to tell the machine what to do. Prolog programs, however, are of the form
>
> This is a fact.
> That is a fact.
> Something else is a fact.
> Etc.
>
> Using Prolog, the programmer does not tell the machine how to do a calculation. The program merely states facts, and the machine automatically finds the facts needed to answer a question. (Biermann 1997: 306–7)

Now, the difference between declarative and procedural languages may seem like a big deal, but the question is whether there is really something different going on down at the level of the circuits of the machine's hardware. There could be differences, particularly in the translation of the higher-level language into assembly language and then machine language, but apart from the translation, is there a difference?

Let's consider the difference between Prolog and Pascal on a simple task like addition. On the surface, there is a significant difference between the two programs. In a procedural language like Pascal, addition looks like this.

```
Z := (X+Y)
```

In a language like Prolog, on the other hand, things look somewhat more complicated. First, we need to write a program to add a list of numbers.

```
f(0, [ ]).
f(S, [X|Y]) := f(Z,Y), S is X+Z
```

What this is saying is first that the function f associates 0 with the null list. It then says that the sum S is found by adding up everything Y but the first entry X in the list [X|Y] to obtain Z. Then X is added to Z to obtain the result. Given these two functions, if we want to add two numbers (say, 7 and 5) we type the following.

```
f(X, [7, 5])
```

Maybe this looks like a lot of trouble. Isn't it easier to just tell the machine to add? Pascal seems so much simpler. But let's return to the Pascal instruction and see what is going on at a lower level.

The line of code Z := (X+Y) is telling the computer to add X and Y, and assign the resulting value to Z, but how is it doing this? First the statement is translated into assembly language, so that the result is something like the following (I'm following Biermann's exposition here).

```
COPY AX,X
ADD AX,Y
COPY CN1,AX
COPY AX,CN1
COPY Z,AX
```

These assembly language instructions are then translated into the binary codes of machine language, so the result might be something like the following.

```
00101101
01001010
00100111
00101111
00100001
```

These instructions are loaded into the computer's memory and are called and used when the instruction pointer of the computer gives their location. So, if 01001010 is called, a particular circuit is activated—let's say that in this case it is a circuit that adds.

The question here is whether there is an interesting difference between the Pascal and Prolog programs at the level of circuitry, and at least as far as the addition operation goes, if the Programs run on the same machine they very well could be calling 01001010 and activating the same circuit.

Where am I going with all this? I submit that computer programs like Prolog and Pascal are fundamentally descriptions of complex physical systems at a level of abstraction that we can understand and use to manipulate the actions of the machine. Some of these languages are at a level of abstraction that make it easy to manipulate the machine to carry out mathematical operations (Pascal being a canonical example) and others make it easy to manipulate the machine to serve as an intelligent data base (Prolog being a canonical example). But apart from the computational resources that are dedicated to the translation of the higher-level language into assembly language and machine language, the basic operations of the machines might well be the same.

Returning to the business of CSAs, the description of the machine in terms of Prolog and the description of the machine in terms of Pascal both satisfy the description of computations in terms of variable substates, connected in the relevant ways. Obviously, both correlate to physical properties of the system, even if the points of correlation are at different levels of physical abstraction. (Clearly, neither program is isomorphic to the operations of at the level of machine language, much less lower level physical properties, so I assume some notion of physical-level-of-abstraction has to be allowed).

What this means for us is that the question of what a data structure is becomes less and less obvious, and the same can now be said for what I have called computational structure. If the very distinction between procedural and declarative programs is called into question, then there isn't much for us to hang our hats on vis-a-vis the machine in isolation. In a bit I'll offer a positive proposal, but first I think it is important to understand just how deep this problem runs: It even raises questions about there being a fact about the machine in isolation that determines the design of the circuitry. Here we take up the arguments from Wittgenstein and Kripke, and we are going to use Fodor as our foil.

Fodor (1975) articulates a view about computation that I believe is still widely held—that the computational states of the physical system just are what they are, quite independently of the external environment. As far as I know, Fodor's view about computation survived his (1994) conversion to externalism about psychological content.

> The physics of the machine thus guarantees that the sequences of states and operations it runs through in the course of its computations re-

> spect the semantic constraints on formulae in its internal language. What takes the place of a truth definition for the machine language is simply the engineering principles which guarantee this correspondence. (Fodor 1975: 66)
>
> ... it is worth mentioning that, whatever Wittgenstein proved, it cannot have been that it is impossible that a language should be private in whatever sense the machine language of a computer is, for there *are* such things as computers, and whatever is actual is possible. (Fodor 1975: 68)

What Fodor is suggesting is that the computational states of the machine are fixed by low level physical properties of the components of the machine and the engineering principles that govern those components. Interestingly, however, Wittgenstein anticipates this move and counters it in *Philosophical Investigations*. Here is how Kripke (1982) framed up the Wittgensteinian response.

> I cannot really insist that the values of the function are given by the machine. First, the machine is a finite object, accepting only finitely many numbers as input and yielding only finitely many as output—others are simply too big. Indefinitely many programs extend the actual finite behavior of the machine. Usually this is ignored because the designer of the machine intended it to fulfill just one program, but in the present context such an approach to the intentions of the designer simply gives the skeptic his wedge to interpret in a non-standard way. (Indeed, the appeal to the designer's program makes the physical machine superfluous; only the program is relevant. The machine as physical object is of value only if the intended function can somehow be read off from the physical object alone). (Kripke 1982: 34)

You might think that, following Fodor, we could figure out the program being run by appeal to the physics of the system and its engineering principles. For example, we could (theoretically) study the logic gates of the computer and determine what it was built to do. But Wittgenstein anticipates this response.

> The machine as symbolizing its action: the action of a machine—I might say at first—seems to be there in it from the start. What does this mean?—If we know the machine, everything else, that is its movement, seems to be already completely determined.
>
> We talk as if these parts could only move in this way, as if they could not do anything else. How is this—do we forget the possibility of

their bending, breaking off, melting and so on? (Wittgenstein 1991 §193)

Like most of Wittgenstein, that is pretty enthymematic, but Kripke (1982) develops this idea in more detail.

> Actual machines can malfunction: through melting wires and slipping gears they may give the wrong answer. How is it determined when a malfunction occurs? By reference to the program of the machine, as intended by the designer, not simply by reference to the machine itself. Depending on the intent of the designer, any particular phenomenon may or may not count as a machine 'malfunction'. A programmer with suitable intentions may even have intended to make use of the fact that wires melt or gears slip, so that a machine that is 'malfunctioning' for me is behaving perfectly for him. Whether a machine ever malfunctions and, if so, when, is not a property of the machine itself as a physical object but is well defined only in terms of its program, as stipulated by its designer. Given the program, once again the physical object is superfluous for the purpose of determining what function is meant. (Kripke 34–5)

So, the idea is that the appeal to the physics of the system has come full circle. You can't determine what program a system is executing until you are clear on whether it is functioning properly, but you can't determine if the system is functioning properly until one has the program that the system is executing.

Let's be clear that we are sticking with talk of human-built computers here and not "natural computational systems." I say this so that we can get the Chomskyans on board here. Chomsky (2000) thinks that the above considerations apply to human-built computational systems but not to natural biological systems.

> Computer models are often invoked to show that we have robust, hard-headed instances of the kind: psychology then studies software problems. That is a dubious move. Artifacts pose all kinds of questions that do not arise in the case of natural objects. Whether some object is a key or a table or a computer depends upon designer's intent, standard use, mode of interpretation, and so on. The same considerations arise when we ask whether the device is malfunctioning, following a rule, etc. There is no natural kind of normal case... Such questions do not arise in the study of organic molecules, the wings of chickens, the language faculty, or other natural objects. (Chomsky 2000: 105)

I doubt that Chomsky can keep the Kripkean skepticism away from biological systems, but it is worth noting that his position marks a retreat from his previous (1986) position, where he was claiming there were narrow facts about computers that determined their computational states.

It might seem like this is getting a bit deep for a paper on virtual tables and sushi, so hang on, because it is about to get deeper. In Ludlow 2019 I argue that the very idea of information and information processing itself is perspectival—meaning that what information is carried by a system depends upon the observer. Others have claimed that information is "subjective." I don't like that formulation, but let's roll with it because it will save me from a long explanation of my proposal and it is clear enough for current purposes. Here is how Galistel and King put it (using the midnight ride of Paul Revere as their example).

> Shannon defined the amount of information *communicated* to be the difference between the receiver's uncertainty before the communication and the receiver's uncertainty after it. Thus the amount of information that Paul got when he saw the lights depends not only on his knowing beforehand the two possibilities (knowing the set of possible messages) but also on his prior assessment of the probability of each possibility. This is an absolutely critical point about communicated information—and the subjectivity that it implies is deeply unsettling. By subjectivity, we mean that the information communicated by a signal depends on the receiver's (the subject's) prior knowledge of the possibilities and their probabilities. Thus, the amount of information actually communicated is not an objective property of the signal from which the subject obtained it! (Galistel and King 2009, Chapter 1)

I like the Galistel and King formulation because it shows why we needn't be skeptics about computation—there is a fact about what is computed and what information is being processed. It just happens to not be a fact in isolation. It is grounded in a broader set of facts about us. What is being computed is a matter of what is legible to us.

Now (finally!) we can get back to our virtual objects. If a computer represents properties of social objects it is by virtue of there being social objects (or types of social objects) out there to be represented. And whether the computer is representing those social objects and properties is a matter of whether it is representing them for us—whether the computations are legible to us, given our social practices and our social ontology. There is nothing about the computer in isolation—no set of narrow properties—that determines those representational states.

4 Virtual (physical) objects as social objects

I hope I made it clear in the previous section that I'm not a skeptic about data structures or computational structures. My point is rather that what those structures are depends in large measure on the legibility of the actions of the machine, and that is going to depend upon the properties that are salient to us and on the kinds of objects that we wish to represent.

Now, in the case of social objects, I think it is pretty easy to make the case that they cannot be *identified* with computational structures. The argument, to some extent, has the form of a multiple instantiation argument—the same social object can be represented in untold ways on untold computational architectures. But this is a bit stronger than a multiple instantiation argument because we know that social objects in virtual worlds do not depend on any kind of computational substrate for their existence. The SSG can not only move to a completely different gaming platform it could also easily operate in RL.

It follows that whatever distributed computational architecture is representing the SSG in a video game, does so only by virtue of their being an SSG out there for it to represent. There is no natural, semi-stable syntactic state or states of the machine that we could point to and say "that is a natural computational object and it is the SSG", nor even "that is a natural computational object independent of the existence of the SSG."

This of course raises all sorts of interesting questions about what social objects are (for an excellent survey of the literature on social ontology see Epstein 2018). On some views, they are grounded in psychological properties, or are socially constructed, or built from social kinds, or bottom out in some diffuse set of physical properties, or some combination of the above. I'm not going to take sides in the debate because I think that what I have to say is neutral with respect to all of these proposals. The only view I have about social properties and socially constructed individuals is that whatever they are, they are not data structures or computational structures.

A view similar to this has been advocated for works of fiction—for example, by Thomasson (2003). On her view, fictional objects are abstract objects that owe their existence to a set of social practices. The difference in this case is that I am claiming that not only do the abstract objects owe their existence to such practices, but that the data structures representing them do as well!

There is also a view out there (sometimes associated with Hegel perhaps) that physical objects too are fundamentally social. That view seems like a stretch for real life chairs and sushi, but I would rather remain neutral on this view as well. What does not seem at all like a stretch is the idea that the virtual objects of Second Life are social objects. If I build a virtual table or virtual sushi in Second Life it is not implausible to think that its being a table or sushi, or really anything, is a function of social consensus. Even if I rez a simple primitive object in Second Life—let's say a cube—one could make the case that it is only an object in Second Life by virtue of some community consensus. Virtual world experiences are full of glitchy things that people agree to ignore. What is special about my cube is that I created it using the agreed-upon protocol for making objects.

This protocol is partly dictated by the company that owns and supports Second Life (Linden Lab), but people have to accept that protocol. Sometimes clever hackers find ways to make objects by flouting the normal protocols. In Second Life, an avatar by the name of Gene Replacement (previously named Plastic Duck) found a way to make "illegally" large primitive objects (theoretically, large objects can mess with the integrity of the server-side computations). These quasi-objects could have been construed as glitches or illegal hacks by the community and they could have petitioned Linden Lab to eliminate those quasi-objects (some did). Instead they were accepted and dubbed "megaprims," and Second Life builders began to incorporate them into their large-scale builds—including one builder who used Gene Replacement's megaprims in the construction of the IBM Pavilion in Second Life (see Mistral 2006). They had community uptake and were thus no longer quasi-objects. They were accepted SL objects.

I think it is even more clear that the virtual *properties* of SL objects is a matter of social consensus as well. My primitive block is (virtually) wooden because we accept it as being (virtually) wooden. We agree that they shall be regarded as virtual wood because of their texture and perhaps sound effects. But many of the properties of the objects in Second Life are quite clearly social properties in other ways—they perform social *functions*.

This is probably the insight that Second Life furniture designers have made a living on for years. People, could just stand around and talk in Second Life, but, for example, when meetings are held (even meetings by IBM) conference rooms are called for and conference tables are called for and people sit at those tables and doing so facilitates discussion. Why? Whether by habit or some feature of human cognitive psychology (or both) tables play an important social function.

The same can be said for the virtual sushi. It plays a social function, just as RL sushi does. Sushi is not designed solely to be an efficient and tasty delivery system for calories and nutrients. It clearly (at times) plays an important social function for those who partake in the meal—even when one is interacting only with the sushi chef. When a friend of mine in Second Life began to make virtual plates of sushi, the intent was to create objects that serve this function.

Now, there is an important difference between virtual social objects like the SSG and virtual physical objects like tables and sushi. The SSG appears to be portable in a way that tables and sushi are not. The SSG can exist in RL, but the table I build and the sushi my friend makes in Second Life cannot be ported into RL. Or so it appears.

The first thing we should note, however, is that under the right conditions my virtual table and my virtual plate of sushi can be ported into another virtual world. Indeed, people have moved much of their inventory from Second Life to another platform called Open Sim.

Here we can get into all kinds of interesting debates about whether a table I make in Second Life is really the table that gets ported over to Open Sim. Perhaps it is only a copy? Notice though that we would encounter similar questions if we started teleporting people and furniture from place to place using Star Trek style teleporters. Is the thing that materializes on the other end the same as the original or is it a copy? The question gets particularly interesting if multiple versions of the original end up on the other end.

I don't have a theory of object identity under teleportation, but I do note that for a single object to "survive" such teleportation, only one object can end up on the receiving end (or in any case, only one of them gets to be the original object), and the original can't stay behind in the original location. One certainly needs some type of causal connection as well—the original has to play a causal role in what appears on the other end.

Similarly, for an object to be teleported from Second Life to Open Sim, the original has to vanish and only one object gets to appear in its place. The original has to be causally related to what appears on the other end. I suppose the other requirement would have to be that migrants from Second Life to Open Sim would have to recognize the table as the very same table. There would have to be social uptake.

Could virtual physical objects in Second Life make an appearance in RL as well? Given the rules of thumb we introduced earlier, certain conditions would have to obtain. The Second Life object would have to go out of existence and there could be only one object appearing in its place. There would have to be a causal relation between the two. These conditions can probably be satisfied for some kinds of virtual physical objects. There used to be (maybe there still are) services that offer to 3D print or mill Second Life objects directly from Second Life. For obvious reasons, this limits the possible portable objects to those that can be 3D printed or milled—plastics, metals, etc. Sushi seems out of the question, given current technology. Suppose we started a service that would print or mill a RL object only on the condition that the SL object was deleted. Suppose further that I furnish my RL house with objects causally related to my (former) SL furnishings in this way. Would we be inclined to talk of these as being the same objects teleported into RL?

I don't know what to say about cases like this, except to say that they soften our inclination to think of virtual physical objects as being importantly different from virtual social objects. In the worst case for my proposal, we would have to think of virtual physical objects as being different because they are intended to mimic certain features of physical objects, including the fact that they can only appear in one location at one time and they may or may not survive teleportation. Such features would not undermine the idea that the objects are inherently social objects; it would simply mean that additional constraints were being placed on the class of objects. The constraints don't make the objects less social; it only allows them to also mimic some features of RL physical objects.

5 Conclusion

Computational data structures and more generally computational structures do not serve as good candidates for the identity of virtual world objects. Nor do they serve as plausible ways of grounding such objects. If I am right, the objects ground the data structures instead of the other way around. In the case of social objects like groups and organizations in virtual worlds this is pretty easy to see. Such objects don't even need computational systems to sustain them. But I think that the same is true for virtual "physical" objects. In the first place, I don't believe there is any single data structure or class of data structures with which a given virtual physical object can be identified. Beyond that, virtual world physical objects look a lot like social objects in a number of ways: They need social uptake to be recognized as objects, their properties require social uptake as well, and their key properties are predominantly social (for example, they instantiate the social properties of a conference table if not the physical properties).

If this is right then understanding the ontology of virtual objects requires an understanding of social objects (and all the squishiness that comes with that). That suggests an interesting opportunity for our better understanding social ontology. Indeed, we could even think of virtual worlds as laboratories for exploring the ontology of social objects, their relation to physical objects, of course how they can be represented by computational systems.[1]

Peter Ludlow
Center for Logic and Epistemology (CLE)
University of Campinas (UNICAMP)
Brazil
peterjludlow@gmail.com

References

Biermann, A. 1997. *Great Ideas in Computer Science*. Cambridge: MIT Press.
Castronova, T. 2005. *Synthetic Worlds*. Chicago: University of Chicago Press.

[1] Thanks to Cory Ondrejka (former Chief Technology Officer of Linden Labs and a principle architect of Second Life) for comments on an earlier draft.

Chalmers, D. 1996. Does a rock implement every finite-state automaton? *Synthese* 108: 310–33.

Chalmers, D. 2017. The virtual and the real. *Disputatio* 9(46): 309–52.

Chomsky, N. 1986. *Knowledge of Language*. New York: Praeger.

Chomsky, N. 2000. *New Horizons in the Study of Language and Mind*. Cambridge: Cambridge University Press.

Clark, A.; and Chalmers, D. 1998. The extended mind. *Analysis* 58: 7–19.

Epstein, B. 2018. Social ontology. *The Stanford Encyclopedia of Philosophy* (Summer 2018 Edition), ed. by Edward N. Zalta, forthcoming URL = <https://plato.stanford.edu/archives/sum2018/entries/social-ontology/>

Fodor, J. 1975. *The Language of Thought*. Cambridge: Harvard University Press.

Fodor, J. 1994. *The Elm and the Expert*. Cambridge: MIT Press.

Galistel, C. R.; and King, A. 2009. *Memory and the Computational Brain: Why Cognitive Science Will Transform Neuroscience*. Chichester: Wiley-Blackwell.

Kripke, S. 1980. *Naming and Necessity*. Cambridge: Harvard University Press.

Kripke, S. 1982. *Wittgenstein on Rules and Private Language*. Cambridge: Harvard University Press.

Ludlow, P. 2019. *Interperspectival Content*. Oxford: Oxford University Press.

Ludlow, P.; and Wallace, M. 2007. *The Second Life Herald: The Virtual Tabloid that Witnessed the Dawn of the Metaverse*. Cambridge: MIT Press.

Mistral, P. 2006. Shock! Banned griefer's tarp covers IBM theatres. *Alphaville Herald*, Dec 15, 2016. <http://alphavilleherald.com/2006/12/ibm_shocker_lls.html>

Putnam, H. 1988. *Representation and Reality*. Cambridge: MIT Press.

Searle, J. 1980. Minds, brains, and programs. *Behavioral and Brain Sciences* 3: 417–57.

Searle, J. 1990. Is the brain a digital computer? *Proceedings and Addresses of the American Philosophical Association* 64: 21–37.

Thomasson, A. 2003. Fictional characters and literary practices. *British Journal of Aesthetics* 43: 138–57.

Wittgenstein, L. 1991. *Philosophical Investigations*. New York: Wiley.