Abstract data types and constructive emergence

Russ Abbott
California State University, Los Angeles
Los Angeles, CA 90032 USA
russ.abbott@gmail.com

Abstract

I suggest that the computer science notion of an abstract data type explains many of the issues raised about the phenomenon known as emergence. The questions in the Introduction to Bedau and Humphreys (2008) are answered.

Emergence as a problem

Broadly speaking, emergence refers to the common observation that many of the phenomena we see around us, although part of the material world, seem to resist description in terms of the laws governing the elements and phenomena from which they derive—and especially that they resist description in terms of the fundamental laws of physics.

In a review of Holland's (1997) *Emergence*, Shalizi (1998) offered the following poetic description.

> Someplace … where quantum field theory meets general relativity and atoms and void merge into one another, we may take "the rules of the game" to be given. But the rest of the observable, exploitable order in the universe—benzene molecules, $PV = nRT$, snowflakes, cyclonic storms, kittens, cats, young love, middle-aged remorse, financial euphoria accompanied with acute gullibility, prevaricating candidates for public office, tapeworms, jet-lag, and unfolding cherry blossoms—where do all these regularities come from? Call this *emergence* if you like. It's a fine-sounding word, and brings to mind southwestern creation myths in an oddly apt way.

When as Shalizi suggests the micro–level consists of the fundamental laws of physics, emergent phenomena are taken to pose a challenge to reductionism. They are regularities in the world that appear to have no direct connection to the regularities of physics.

In a well-known early paper, Fodor (1974) cited Gresham's law as such a regularity. Although he didn't use the term *emergence*, his point was that the regularities characterized by the special sciences—economics in this case—were autonomous from those identified by physics. What is one to make of this? In the remainder of introduction I'd like to discuss some preliminary issues.

Does emergence require regularities?

Among the central concerns of both Shalizi and Fodor is that emergent phenomena reflect a regularity. Fodor is particularly emphatic about it. In a 1997 follow-up to his 1974 paper, he focuses explicitly on regularities.

> The very *existence* of the special sciences testifies to reliable macro-level regularities … Damn near everything we know about the world suggests that unimaginably complicated to-ings and fro-ings of bits and pieces at the extreme *micro*-level manage somehow to converge on stable *macro*-level properties.

On the other hand, neither Bedau and Humphreys (2008) nor O'Connor and Wong (2009) imposes a similar requirement—at least not explicitly.

Bedau-Humphreys:

> Emergence relates to phenomena that arise from and depend on some more basic phenomena yet are simultaneously autonomous from that base.

O'Connor-Wong:

> [Emergent] entities (properties or substances) 'arise' out of more fundamental entities and yet are 'novel' or 'irreducible' with respect to them. … Each of the quoted terms is slippery in its own right.

All that either requires is that to be emergent a phenomenon (or entity, property, or substance for O'Connor and Wong) must "arise" from more basic phenomena (or entities, properties, or substances), not that it exhibits any sort of regularity. That raises the question as to whether regularity is required for something to be labeled emergent.

Although Bedau-Humphreys and O'Connor-Wong do not use the term "regularity," they both use other terms that have similar implications.

Bedau-Humphreys requires that emergent phenomena be "autonomous" from their base. But for something to be autonomous implies that it

behave in a way that can be described independently of the elements from which it "arises." That description constitutes a regularity.

O'Connor-Wong requires that for something to be emergent it must be somehow 'novel' or 'irreducible' with respect to the elements from which it arises. Again, novelty requires a description of how the emergent element is novel, which means that the novelty must be described in terms that are not just an enumeration of the component elements. Irreducibility requires a description of the irreducible feature. By any definition of irreducible that description must be given in terms other than an enumeration of the components.

Given all this, it seems to me that Shalizi and Fodor are on the right track in requiring regularity as necessary for emergence. In fact it seems to me that one of the fundamental features of emergence is the existence of non-primitive regularities. So from here on I will assume that a central aspect of emergence is that emergent phenomena exhibit some sort of regularity.

Regularity and abstraction

Chaitin (2009) credits Leibniz with having originated the idea that a phenomenon exhibits a regularity if and only if it can be described in a more compact way than by a brute force enumeration of its components.

> [In Section VI of the *Discours de métaphysique* (1686)] Leibniz considers "experimental data" obtained by scattering spots of ink on a piece of paper by shaking a quill pen. Consider the finite set of data points thus obtained, and let us ask what it means to say that they obey a law of nature. Well, says Leibniz, that cannot just mean that there is a mathematical equation passing through that set of points, because there is always such an equation! The set of points obey a law only if there is a **simple** equation passing through them, not if the equation is "fort composée" = very complex, because then there is always an equation.

Chaitin's Algorithmic Information Theory is essentially a formalization of that insight: the complexity of a phenomenon is the length of its shortest description.[1] A regularity, then, is characteristic of something that is not maximally complex, i.e., something that can described in more compact terms than a brute force enumeration of its elements.

Fodor's (1974) example of Gresham's law—bad money drives out good—provides a good illustration. As Fodor noted, it isn't necessary to specify what a particular economy uses for money. What matters is that there is a regularity that applies to monetary exchanges.

> Some monetary exchanges involve strings of wampum. Some involve dollar bills. And some involve signing one's name to a check. … The point is that monetary exchanges have interesting things in common; Gresham's law, if true, says what one of these interesting things is.

Another way of putting this is that a regularity is something for which there is an abstract characterization, one that doesn't depend on the details of the elements described.

So where does this leave us? For a phenomenon to be emergent requires that it have an abstract characterization. This was the basic message of (Abbott 2006). I'll return to the issue of regularities when discussing entities.

Is emergence mysterious?

As I'll explain below, I don't find emergence mysterious. But a number of philosophers—including Bedau, Fodor, and Humphreys—have been quite open about the fact that they do. Here are some extracts that illustrate their puzzlement.

The final section of (Fodor 1997), the section from which the preceding extract is taken, is entitled "**Conclusion (molto mysterioso)"**. Referring to his statement that "*micro*-level [interactions] manage somehow to converge on stable *macro*-level properties", Fodor writes the following[2].

> The "somehow" really is entirely mysterious. Why should there be (how could there be) macro-level regularities *at all* in a world where, by common consent, macro-level stabilities have to supervene on a buzzing, blooming confusion of micro-level interactions. …

[1] A consequence of that definition is that a collection of random phenomena is *maximally* complex because no compression is possible. Since the phenomena are random there is no rule or regularity that describes them other than their enumeration.

[2] The extract is slightly paraphrased. The complete extract is provided in the Appendix.

> So, then, *why is there anything except physics*? … I expect to figure out why … the day before I figure out why there is anything at all. …

Bedau and Humphreys were sufficiently puzzled by emergence to produce the previously-mentioned edited collection of papers. The following extracts are from the Introduction to that volume.

> The very idea of emergence seems opaque, and perhaps even incoherent.
>
> When we finally understand what emergence truly is, we might see that many of the [supposed] examples are only apparent cases of emergence. Indeed, one of the hotly contested issues is whether there are *any* genuine examples of emergence.
>
> [T]he proper characterization of emergence still is contested.
>
> The study of emergence is still in its infancy and currently is in a state of considerable flux
>
> [M]any contemporary philosophers think that emergence is a rare and special quality found only in extremely distinctive settings, such as human consciousness. Others think that emergence is quite common and ordinary.
>
> For [philosophers] who think that nothing is truly emergent, the question still arises whether this state of affairs is simply an accident or whether the very idea of emergence is incoherent.
>
> Is emergence an objective feature of the world, or is it merely in the eye of the beholder?
>
> In what ways are emergent phenomena autonomous from their emergent bases, … irreducible to their bases, inexplicable from them, unpredictable from them, supervenient on them, multiply realizable in them?
>
> Does emergence necessarily involve novel causal powers, especially powers that produce "downward causation" in which emergent phenomena have novel effects on their own emergence base.
>
> Emergence … is simultaneously palpable and confusing.

Bedau, Fodor, and Humphreys are not the only philosophers who acknowledge being puzzled by emergence. But my purpose is not to list all philosophers who have made such an acknowledgement. It is simply to establish that emergence has been—and apparently continues to be—an open and apparently somewhat puzzling philosophical problem.

The basic questions regarding emergence

Can we say what the fundamental philosophical issues are regarding emergence? Much of the Bedau-Humphreys Introduction is devoted to formulating what they take to be the important philosophical questions. Later in this article I respond to those questions.

I'm not sure, though, that any of their questions quite get to the fundamental issues. To explain why I say that, I'd like to turn to the Game of Life[3], which has become something of a white mouse for thought experiments about emergence.

If one runs the Game of Life from a random starting configuration, various patterns appear. One of the best known is the glider, a sequence of configurations such as the following.

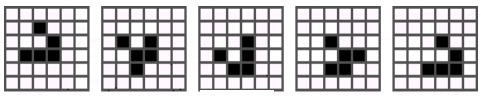

Figure 1. A glider

As the illustration shows, after four time steps the glider pattern reconstitutes itself one square to the right and one square down.[4] When a glider appears during a Game-of-Life run, it looks like a little creature scurrying across the grid.

The glider is a nice example of emergence. It is a macro-level regularity that arises from the micro-level phenomena of Game-of-Life cells going on and off according to the Game-of-Life rules. It is a macro-level regularity in that it has properties that can be described abstractly. For example, a

---

[3] Probably most readers are familiar with the Game of Life. But to review, the Game of Life is an unbounded, totalistic (the action taken by a cell depends on the *number* of neighbors in certain states—not on the states of *particular* neighbors) two-dimensional cellular automaton. It operates in discrete time steps. Each cell is binary. At any moment, a cell is either "alive" or "dead"—or more simply on or off. At each time step the following rules are applied simultaneously to each cell to determine whether it will be alive or dead at the next time step.

> A live cell stays alive if it has two or three live neighbors; otherwise it dies.
>
> A dead cell with exactly three live neighbors becomes alive.

[4] Gliders may move in any of the four diagonal directions. This one moves east-south-easterly.

glider moves across the grid; it has a size; and its motion can be characterized by a velocity, i.e., a speed and a direction. These statements characterize a glider as an abstract phenomenon. They do not rely on how gliders are implemented by the Game-of-Life rules, i.e., which individual cells are going on and off. In fact, the Game-of-Life rules don't talk about things moving across the grid at all. All they talk about is cells going on and off.

There is certainly nothing mysterious about how a glider comes to be. As Figure 1 shows, each configuration follows from the preceding according to the Game-of-Life rules.[5]

So what is the emergence issue? Is it, as Fodor suggests, why the Game of Life gives rise to any regularities at all? Why isn't every Game-of-Life run just a "buzzing, blooming confusion" of cells going on and off?

That seems similar to asking why there is mathematics. Mathematics is the study of (often unforeseen) regularities. One could just as well ask why there are more real numbers between 0 and 1 than there are integers, or why every positive integer can be expressed as the sum of 4 or fewer squares—e.g., $23 = 3^2 + 3^2 + 2^2 + 1^2$—or why "Fermat's last theorem" is true, or why any other mathematical theorem holds.

The mathematical answer to why a theorem holds is given by the proof of the theorem. Similarly the question "Why are there gliders?" is answered by Figure 1, which shows why there are gliders—or at least how gliders come to be.

Given any emergent phenomenon, it seems to me that its *how* question—how does this phenomenon come to be—is (or will be[6]) answered by science.[7]

But if the *how* question is not the central issue, are there other basic questions? The following are what I take to be the fundamental questions about emergence—along with brief versions of my answers.

1. *What—if anything—characterizes what we think of as emergence?*

   My answer. Emergence pertains to macro phenomena that exhibit regularities. These are phenomena that may be characterized abstractly, i.e., structurally or functionally at the level of the phenomena themselves and without reference to the properties of the phenomena from which they derive. In the case of the glider, for example, part of its abstract characterization is its velocity.[8]

2. *Is there a useful categorization of emergent phenomena in terms of how they come to be?*

   My answer. Energy, as characterized by physics, is the common material basis for all *material* emergent phenomena.[9] An analysis of

---

[5] In philosophical terms, Figure 1 provides what I understand to be a reductive explanation for how a glider is related to the rules of the Game of Life.

[6] Consciousness is frequently cited as emergent. We can't answer the how question about it yet. But I'm convinced that we will be able to eventually.

[7] This is true even if the regularity is multiply realized. The fact that a regularity can be realized in multiple ways doesn't mean that we don't understand how each one of those ways produces the regularity. In general we do.

(Abbott 2006) includes a characterization of Game-of-Life patterns. It is possible to specify how such patterns interact in terms of the patters themselves and without reference to the Game-of-Life rules. For example, one of the early patterns was a "glider gun" (see the "Gosper glider gun" page in the *Life wiki* 2010), which consists of four patterns. Two move horizontally (or vertically) back and forth between the other two, which act as reflecting barriers. When the moving patterns meet in the middle, they bounce off each other and head back toward the barriers. In addition, each collision and rebound generates a glider, resulting in a stream of gliders moving away in a diagonal direction.

What is significant about this description is that the interaction of the patterns can be expressed strictly at the pattern level and without reference to the Game-of-Life rules. Many such pattern interactions are described in the Life wiki. See http://conwaylife.com/wiki.

[9] In what may cause some confusion, energy is not relevant for emergence in the Game of Life or other computational frameworks. The reason is that computational environments are energy subsidized. Elements within the framework do not have to worry about where the energy comes from that creates them and keeps them in existence. The Game of Life, for example, turns cells on and off without consideration of the energy that would be required to do that were the Game of Life implemented physically.

the role that energy plays in physical emergence leads to a natural division of those phenomena into two categories.

- Some emergent phenomena—like molecules and solar systems—exist in what are known as energy wells. Energy is required to pull them apart—to destroy them.
- Other emergent phenomena—like biological organisms and social organizations—require a continual supply of external energy to keep themselves together. But these phenomena are organized in such a way that they are able to extract enough energy from their environment to maintain their form and functionality.

I call the former class of entities static and the later class dynamic. See Table 1 and the section below on emergence and entities. The key in both cases is how energy is used to create or sustain the emergent phenomenon.

3. *Is there anything common to all emergent phenomena?*

   My answer. Emergence results from the imposition of constraints. Emergence occurs when constraints are imposed on elements in such a way that the constraints persist under the conditions in which the constrained elements finds themselves. In the case of static entities the constraints are imposed and maintained by the fundamental laws of physics. In the case of dynamic entities, the constraints depend on the structure and functioning of the entity.

I don't have good answers to two follow-up questions.

4. *What makes it possible to impose constraints?*

   This question asks whether there is a general characterization of *the means* that are—or will ever be—available to impose constraints. What are the mechanisms through which constraints can be imposed? For static entities the answer is that constraints are imposed by combinations of the elementary forces of physics. For dynamic entities, I see no similar characterization.

That's why I wrote that energy is the common basis for all *material* emergent phenomena. This issue is explored further later in this paper.

5. *How might one characterize the kinds of constraints that can be imposed?*

   This question asks whether there is a general characterization of *the form* constraints may take. Other than to say that a constraint reduces entropy, I don't see a general characterization of the forms that constraints may take.

Is that all there is?

Having reached this point it strikes me that there may be a sense of let-down. Is emergence just putting things together in a way that they will stay together? Is it no more profound than that?

For the most part I think that's true. Emergence is not that complex. It's basically a common-sense notion. Putting things together often produces new things with new properties and new functions. As builders and as organizers, human beings have been putting things together to get new things for millennia. Nature has been putting things together and getting new things for even longer.

Perhaps a little bit more

There is one aspect of how constraints work that may add to our understanding of emergence. That is the realization that organizational structures and processes need not be specified or applied externally or explicitly. They may "arise" from pre-existing structures and processes.

The Game-of-Life glider (as usual) provides an illustrative example. There are no "glider constraints" that one imposes from the outside to create a glider. Nothing is added to the Game of Life to hold cells together in a glider pattern. Everything required to make a glider is intrinsic to the Game of Life.

So where does a glider come from? The glider results from particular arrangements of on and off cells—it "emerges" when one of the glider configurations appears on a Game-of-Life grid. As Figure 1 illustrates, the glider pattern is self-perpetuating. If any of the 16 cell configurations[10] that produces a glider happens to occur on a Game-of-Life grid, the ensuing sequence of patterns will be a glider—which will, unless interfered with, perpetuate itself indefinitely.

[10] There are 16 patterns because (a) a glider consists of a sequence of 4 patterns and (b) gliders can be oriented in any of four directions.

What may seem striking—and what people often find fascinating about mergence—is that a glider, a phenomenon that embodies a regularity that isn't apparent in the Game of Life rules, can appear within the Game of Life without any additional structure imposed from the outside.

If no additional structure is imposed, where did the glider's regularity come from? It didn't come from outside; yet it wasn't there before. Is that a source of mystery?

Well, as I suggested before, one can ask the same question about mathematical theorems. They are (seemingly additional) structures that appear within a framework of axioms even though no new structure is imposed.

In mathematics, of course, the structure was there all the time. It was implicit in the axioms. In emergence—at least emergence related to dynamic (and computer generated) entities—the story is a bit more open-ended.

Harnessing and channeling processes and energy flows

Even though emergence is not, in my opinion, mysterious, and even though emergence is little more than putting things together so that stay together, a full understanding of emergence requires that we think of the world in a way that some may find unfamiliar. Because understanding emergence requires this new perspective, it may sometimes seem that emergence comes from out of nowhere. What is this new perspective? It is to see the world in terms of processes and energy flows.

The physical mail system—for those of us who remember it—offers an illustration. When one puts a letter into an envelope, addresses and stamps the envelope, and then puts the envelope where it will be picked up by the mail system, one is making use of an ongoing process. When one drops the envelope into that process, one expects the process to carry it away and eventually to deliver it to its designated destination.

The thing to keep in mind in this picture is that there is a pre-existing ongoing process—the operation of the mail system—and that we make use of that process to accomplish an objective.

If one thinks about it a bit more carefully, one sees that not only is this a matter of dropping an item into a process, there is also an element of harnessing and channeling that process. The mail system has built into itself a way (a) to read addresses and based on those addresses (b) to direct the addressed items into various of its internal flows as the item moves through the system. When it works correctly—as it does most of the time—the result is that items are delivered to the addresses they bear.

When one puts an address on an envelope one is programming the mail system—although few people think of themselves in this way. The address on an envelope makes use of the fact that the mail system includes an interpretive function. One makes use of that interpretive function to harness and channel the process in the service of having the envelope delivered to its intended destination.

This way of thinking—producing a result by harnessing and channeling an existing process—may seem unfamiliar. But for software developers it's our daily occupation. Software development is the creation of symbolic structures that harness and channel the ongoing operation of an underlying computational engine in the service of producing a desired result. Programmers put computer instructions together in such a way that when the computational engine encounters them a desired result occurs.

Harnessing and channeling an existing process is not limited to human activity; it's commonplace in nature as well. Perhaps the most widely known example is the way some plants make use of bees for pollination.[11] Although not exactly like our mail system, plants take advantage of bee activity (the underlying process) to have pollen delivered from male to female.

(Margulis and Sagan 1986) famously put this kind of interaction among organisms more elegantly. "Life did not take over the globe by combat, but by networking." Many organisms depend on the activities of other organisms for their own survival and reproduction.[12]

[11] I understand that the apparent teleological flavor of the preceding may seem inappropriate. I explain my perspective in the section on mechanism, function, and purpose.

[12] For another example, see the discussion of the liver fluke below.

Recent discoveries about moth migration illustrate how biological organisms also make use of non-biological processes. The following is paraphrased from the abstract of (Chapman and Hill 2010).

> [The] ability to select seasonally favorable, high-altitude winds [and to correct to a significant degree for crosswind drift] is widespread in large day- and night-flying [insects]. [These] behaviors increase migration distances by 40% [and] match the sophistication [of] migrant birds.

This is from the associated institutional press release.[13]

> [The] insect migrants have a compass sense that enables them to select winds which will take them in their chosen direction, and to travel at speeds of up to 100 km per hour.

In other words, moths are not just blown willy-nilly but select which winds they will ride as they migrate between their summer and winter habitats. This illustrates once again how existing processes can be harnessed and channeled to achieve an end.

How is the harnessing and channeling of processes related to energy flows? Processes are activities; they cannot proceed unless they are driven by energy. Every process reflects an energy flow.

Although all processes depend ultimately on energy flows, we typically (and unfortunately in my opinion) don't think of the processes defined by automata—such as the Game of Life or a Turing machine—in terms of energy flows. We think of automata as simply operating according to their rules. They do what they do because that's how they are defined. Energy is not an issue.

Furthermore, the elements that "emerge" as a result of those processes need not be concerned about energy either. If an instance of the Game of Life were suddenly to stop—for whatever reason—the gliders within it wouldn't be expected to get it started again. It is simply assumed that automata will continue to operate as defined—and if they don't it isn't a concern of the emergent elements that inhabit the virtual worlds enabled by those processes.

[13] http://www.rothamsted.ac.uk/Research/Centres/PressReleases.php?Centre=&PRID=87

Of course that's not true in the real world. If the source of energy for a biological organism disappears, it either finds another source, finds a way to live without the energy or services it has received from that source, or it dies.

Too often we ignore the fact that a great deal of how we live has to do with manipulating energy flows. Consider something as simple as turning on a light. It is not the flipping of the switch that causes the light to go on. Flipping the switch enables a harnessed energy flow (an electric current potential) to move along a temporarily blocked channel through the light bulb causing it to produce light. Most people, unless you pressed them, don't think about it this way.[14]

Perhaps one reason we do not conceptualize the world as a dynamic intermingling of energy flows is that we have tamed and domesticated the external energy flows that we use most often—electricity and gasoline are the primary ones. We now think of them more in their state of potential energy than as ongoing processes. And even when we release their potential and use them as energy flows—e.g., when a light is on—we still don't think of ourselves as riding an energy flow. We tend to see ourselves as in a static situation one of whose properties is the existence of light supplied by a light bulb. We think of flipping the switch simply as toggling the state of the light between off to on.

The computer science approach to emergence

In Computer Science we tend not to use the term *emergence*. But the preceding discussion has hinted at the computer science equivalent. In computer science we think about abstractions and how (or whether) they can be implemented. The term we use for such abstractions is *abstract*

[14] The (Pearl 2000)/(Woodward 2003) approach to causality encourages the sort of thinking that attributes causality to switch-flipping by focusing on interventions as causes.

In my view the more fundamental question about interventions should concern where the energy that powers the intervention act itself comes from. We should be asking where the energy that we use to flip the switch comes from and how that energy flow interacts with the other energy flows in our lives.

data type. It is my suggestion that once one understands the notion of an abstract data type questions about emergence are easily answered.

Computer Science uses the term *type* (or *data type*) for what in philosophy is called a *kind*. An *abstract* data type is a type that is characterized exclusively in terms of specified properties. What this means is that the properties of an abstract data type are specified abstractly and not in terms how the abstract data type may be realized.

Sometimes these properties are expressed as a set of axioms. A nice example is Peano's well-known axioms for the non-negative integers[15]. Any concrete realization of these axioms will have the properties of the non-negative integers. The properties inhere in the abstraction and not in the realization.

More often the properties of an abstract data type are expressed informally—but clearly enough that there is general agreement about what the specification means. The Turing machine is a nice example. The functionality of a Turing machine is well defined. Yet Turing did not provide a Peano-like set of axioms when he defined a Turing machine. His description was rigorous enough that no one has argued that the specification is unclear or ambiguous.[16]

It's important to realize that Turing machines and Peano arithmetic are self-contained abstractions. Unlike emergent phenomena they are not defined as "macro" properties that have somehow arisen from some "micro" phenomena. They are simply a set of self-consistent properties and functions. This is generally the case with abstract data types.

Another way to put it is that abstract data types are pure imagination. They enable software designers to introduce new concepts. An abstract data type is an imagined category of things that behaves in an imagined way.

That is not to say that abstract data types are arbitrary. They are defined for specific reasons. Turing defined the Turing machine the way he did—rather than in some other way—because it had properties he thought were important.

More frequently abstract data types are defined to have properties that are patterned after non-software elements with similar properties. Peano developed his axioms as a way to formalize what he thought were the essential features of the non-negative integers.

For a more commonplace example consider a word processor. Word processors include a number of familiar abstract data types. Among them are the *word*, the *paragraph*, the *page*, and the *footnote*. One of the criteria we apply to word

[15] Here are Peano's axioms as presented in Weisstein (2009).

> Zero is a number.
>
> If ***a*** is a number, the successor of ***a*** is a number.
>
> Zero is not the successor of a number.
>
> Two numbers of which the successors are equal are themselves equal.
>
> (Induction axiom.) If a set ***S*** of numbers contains zero and also the successor of every number in ***S***, then every number is in ***S***.

[16] Here is Turing's (1936) introductory description of what has come to be called a Turing machine. Turing added formalization later in the paper, but as far as I can tell, he never provided an axiomatic basis for his machine.

> We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions q_1, q_2, ..., q_R which will be called "*m*-configurations". The machine is supplied with a "tape", (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the *r*-th, bearing the symbol S(*r*) which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its *m*-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the *m*-configuration q_n and the scanned symbol S(*r*). This pair q_n, S(*r*) will be called the "configuration": thus the configuration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (i.e. bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the *m*-configuration may be changed. Some of the symbols written down will form the sequence of figures which is the decimal of the real number which is being computed. The others are just rough notes to "assist the memory". It will only be these rough notes which will be liable to erasure.

processors is to ask how well the properties of their abstract data types parallel those we intuitively associate with "kinds" of the same name that we understand from other contexts. Is a *word* in Microsoft Word like what we understand a word to be in English?

Contrast a word processor with a simpler so-called text-editor (like Microsoft Notepad) that enables users to manipulate text but that implement far fewer data types. Microsoft Notepad has no data types corresponding either to the paragraph or the footnote.

A collection of interacting abstract data types is known as a *level of abstraction*—which in some ways is analogous to a special science. Like an abstract data type, a level of abstraction is defined in terms of its abstract components. It too is therefore a fantasy world—one defined in terms of itself—but like so many fantasy worlds, motivated by similar real-world collections of kinds.

The level-of-abstraction implemented by a computer program is generally known as a *conceptual model*. To use a piece of software effectively one must first master its conceptual model—understand the types it implements and the operations it enables users to perform on elements of those types.

I want again to emphasize that each of these—an abstract data type, a level of abstraction, and the conceptual model implemented by a piece of software—is generally defined self-referentially. Each consists of a self-contained world defined (circularly) in terms of its own elements.

An example of a sophisticated abstract data type is the *word* abstract data type in Microsoft Word. When one *selects* a *word* in Microsoft Word (by double clicking within it) one gets the symbols that make up the English word along with the space (if any) that follows it. Imagine then pasting the copied *word* in two different places: one at the end of a sentence where the pasted *word* is followed immediately by a period and one in the middle of a sentence where the pasted *word* was preceded and followed by another *word*. Microsoft Word treats the space selected with the word properly in both cases. In the first case no space is left between the pasted *word* and the following period. In the second, spaces are left between the pasted *word* and both the preceding *word* and the following *word*. Try the same thing in Microsoft Notepad, for example. You will see that Microsoft Notepad is not nearly as sophisticated in its treatment of the analogous abstract data type.

Notice that in none of the above was it necessary to talk about *how* abstract data types are implemented. I didn't say whether a *word* is stored as bytes in the computer's memory or which internal computer operations are performed when one copies and pastes a *word*.

A *word* is a sequence of *symbols* (i.e., a letter, digit, punctuation character, etc.), but users neither need nor want to know that a *symbol* has for a long time been stored in the computer's memory as a byte. They almost certainly don't want to know that now that word processors deal with symbol sets that have more elements than there are distinct bit configurations in a byte (such as symbols sets that include characters from a wide range of alphabets), a symbol is no longer stored simply as a byte. For users, a symbol is a symbol. It should have the properties of a symbol as an abstract data type and not properties that depend on whether or not it is stored as a byte.

A non-abstract data type

Not all computer data types are abstract. Consider the collection of floating point numbers and their associated arithmetic that is implemented by a particular computer—especially a computer of a few decades ago before floating point numbers were standardized. In this case there is no independent specification. The implementation *is* the specification. The data type consists of the numbers that can be represented and the operations that the computer can perform on them. This is a good example of a non-abstract data type.

One of the issues raised about emergence is autonomy. Emergent phenomena have properties that are said to be autonomous of their base. Another issue is multiple realization. It is often argued (or more frequently just assumed) that if a property is multiply realized it must be autonomous. The preceding example of a non-abstract data type can be used to show that multiple realizability is not a definitive demonstration of autonomy.

Imagine software that emulates the original computer but that runs on a modern computer. The emulating software would implement the original floating point data type. Is that data type now

autonomous? It is clearly *not* autonomous of its original implementation; its properties were defined by its original implementation. Yet those properties serve as an abstract (autonomous) specification for the emulator. So the data type is autonomous of one of its implementations and not autonomous of another.

Furthermore, once the emulator is written the data type would be multiply realized. That of course would not change the fact that the properties and functionality of the data type are not autonomous of the computer where it first appeared.[17]

Type theory

In the preceding I've tried to provide an intuitive feeling for abstract data types. I don't want to leave the impression that that's all there is. Type theory—introduced by Bertrand Russell to prevent Russell's paradox—has developed over the past century in multiple directions—including mathematical logic and the theory of programming languages—to become quite sophisticated.[18] But no matter how complex type theory becomes, it is always the case that type definitions are self-contained; types are not defined in terms of their implementations or realizations.

Summary comments about abstract data types

I hope that the preceding has conveyed the idea that abstract data types are defined independently of their implementation. Furthermore, abstract data types may be implemented (or realized) in multiple ways. For example, the abstract data types implemented by Microsoft Word are echoed to a great extent by the abstract data types implemented by Open Office, the open source competitor. Yet the Open Office implementations of those types are necessarily different from the Microsoft Word implementations—or Microsoft would sue the Open Office developers for copyright infringement.

Multiple realizability of an abstract data type is neither required for nor a demonstration of the autonomy of an abstract data type. It's the reverse that's true: multiple realizability is enabled by the fact that the type has an (autonomous) abstract specification.

The fundamental point, though, is that the notion of an abstract data type requires a strict separation of specification from implementation. The specification characterizes the properties of the type; the implementation/realization embodies those properties in something concrete.

The notion of an abstract data type has become one of the foundational elements of computer science and is now part of the standard lower division computer science curriculum.

Emergence as a fact

Many things in the world can be described abstractly—in terms of regularities that are not directly related to the fundamental elements of physics. These are the regularities to which Fodor (1997) was referring when he wrote "The very *existence* of the special sciences testifies to reliable macro-level regularities." After making the

---

[17] There is an old joke in the software community about letting the implementation define the abstraction. If one finds a bug in a program, instead of fixing it one should change the user manual—thereby changing a bug into a feature.

[18] For example, types need not be just simple classes of like items. One type may be a subtype of another: *human being* is a subtype of *mammal*, which is a subtype of *animal*, etc. Types can also be defined functionally: *male* cuts across the types just mentioned.

There are also polymorphic (or structural or parameterized) types (like *list*) in which a structural regularity is understood to be a (parameterized) type whose properties are independent of the type of thing stored in the structure. A list of numbers and a list of names are both *lists*, and have list-properties.

Functions too (including, intuitively, operations and processes) can also be understood as having a type. The process of counting the number of people standing in a line derives a number from a queue. The type of such an operation/function is one that maps queues (no matter what sorts of things are in the queue) to non-negative integers. That type would be written as something like this.

queue(t) → Number

The "t" in parentheses means that the queue type is parameterized by the type of its members, which can be anything. Treating functions as typed objects is especially important in modern programming languages.

case for emergence as a mystery, (Fodor 1997) ends with the following.

> Lots of different sorts of micro-interactions manage, somehow or other, to converge on much the same macro-stabilities. The world, it seems, runs in parallel, at many levels of descriptions. You may find that perplexing; you certainly aren't obliged to like it. But I do think we had all better learn to live with it.

Learning to live with it means accepting the existence of naturally occurring regularities—taking those regularities as a fact—a starting point. That's the approach I recommend. In doing so one converts the special sciences from a problem that needs to be explained into an ally. Instead of asking how there can possibly be macro-level regularities I take the existence of macro-level regularities as given and asking what that says about the world.

One thing it says is that many features of nature can be characterized abstractly. Any such description is the specification of an abstract data type. I suggest then that in the world of material phenomena emergence is best understood as the realization/implementation of an abstract data type.

It is important to realize, though that nature is *not* capable (as far as I know) of creating an abstract data type without realizing/implementing it. That is, nature cannot create a *specification* of an abstract data type.

Human beings create unrealized specifications all the time. When designing computer programs, software developers specify the abstract data types they intend to use. These specifications exist even though no instance of those types have been created—and even though designs for how those data types will be implemented may not have been developed.

Specifications are created in other fields as well. A constitution is a specification for a government before the government is created. Similar documents provide abstract descriptions for corporations, clubs, and other social organizations. *Robert's Rules of Order* (e.g., Robert 2000) provides a specification for meetings as an abstract data type.

But nature—at least non-human nature—can't do this. There is no way for nature to create a specification of an abstract data type.[19]

The fact that stand-alone specifications don't exist in nature may be one of the reasons it has been difficult to understand emergence. My view is that emergence occurs in nature in the form of realized abstract data types. Yet it is the fact of the abstraction—that one can provide an abstract specification—and not the realization of that abstraction that establishes a phenomenon as emergent. The fact that abstractions don't themselves exist as independent physical elements may make it difficult to think of them as playing a role in emergence.

Emergence and entities

The primary notions in the version of emergence I am proposing are (a) abstract data types and (b) instances/implementations/realizations of abstract data types. Since nature does not create unrealized abstractions, it is the instances that come into existence. An instance of an abstract data type is an entity, a thing. Consequently entities are central to my version of emergence.[20]

As far as I can tell, there is no philosophical consensus about what one means by the term *entity*. One common view is that the entities are whatever scientists talk about. In explaining Quine's approach to ontology, Smith (2003) put it this way.

> Quine takes ontology seriously. His aim is to use science for ontological purposes, which means: to find the ontology in scientific theories. Ontology is then a network of

[19] Even a string of DNA—the closest nature comes to representing a description of an entity in a form other than the entity itself—is not a specification of the functionality of a biological organism. The DNA does not describe the organism's properties or functionality. It is not a specification.

Neither is DNA a design for an organism. If anything it's more like a parts list along with control information for the order in which those parts are to be made available. Although some of the parts play important roles in the mechanism that controls the order of production of other parts, DNA has little to say about what use is made of most of the parts after they become available.

[20] I discussed entities in both (Abbott 2008) and (Abbott 2009).

> claims, derived from the natural sciences, about what exists coupled with the attempt to establish what types of entities are most basic. … This is defined by the vocabulary of the corresponding theory and (most importantly for Quine) by its canonical formalization in the language of first-order logic. …
>
> His so-called 'criterion of ontological commitment' is captured in the slogan: *To be is to be the value of a bound variable.*

That's fine. The network of claims made by a science about what exists is essentially the level of abstraction or conceptual model defined by that science. But that doesn't tell us the sorts of things to which scientists tend to make ontological commitments.

The simple answer, in my view, is that science looks for and attempts to understand and characterize regularities. Regularities are the entities of science.

So what is a regularity? As we said above it is something that can be described abstractly, i.e., more concisely than a simple enumeration of its components. Another way to put it is that an entity is a realized abstraction, a persistent pattern—some aspect of the world that has lower entropy than its surroundings. In my view then, entities are realized abstractions or persistent patterns.

The Game-of-Life glider (yet again) provides a nice illustration. Presumably one wants to think of gliders as entities. Yet gliders are *not* collections of particular grid cells. Gliders are *patterns* that propagate across the Game-of-Life grid.

Mechanism, function, and purpose

A fundamental question with regard to entities concerns the relationship between structure and function. An abstract data type is defined by its functionality; it's implementation is not relevant. But an entity is an implemented abstraction; the implementation does matter. So what is the relationship between function and structure?

Mechanism

The term *mechanism* is surprisingly difficult to define. I would define a mechanism as a collection of elements whose interactions are well understood and whose workings can be predicted in terms of that understanding. The sorts of mechanisms I have in mind are any of the standard models of deterministic computation or other intuitively mechanistic constructions.[21]

An important property of a mechanism is that it characterizes the *operation* of an entity. It has nothing directly to do with how the entity interacts with or functions in the world. A mechanism is blind to its functionality. A mechanism simply does whatever it does in a straight-forward mechanical way.

Intuition becomes cloudy, however, when one considers mechanisms that respond to external signals. A simple (and by now very well worn) example is a heater with a thermostatic controller. Such a mechanism produces heat until its temperature sensor reaches a set level—at which point its heating process ceases—until the sensor cools off.

I resisted the temptation to write that the heater continues to heat until *it senses* that the temperature of the surrounding air has reached a certain level, at which point *it turns off* the heating element.

I resisted that temptation because the sensor doesn't *actively sense* the air temperature in the common understanding of that term. It doesn't put its finger in the air and say, "Feels like about 72 degrees Fahrenheit." The sensor element—typically a bi-metallic strip—is such that it changes shape as the temperature changes. Like all other aspects of mechanisms, that change is blind. It just happens because of the physics of the materials.

Similarly, the sensor doesn't *actively turn off* the heating element in the common understanding of that term. It's not as if it says to itself, "Now that the room is a comfortable temperature, I'll turn off the heating element." It's much more direct and mechanical than that. The very change in the sensor shape breaks a circuit that is required to be unbroken for the heating element to be on—at which point the heating element stops heating.

The only reason the sensor can be said to perform functions such as "sensing the air" and "turning off the heating element" is that the mechanisms we built into it *result in* those functions being performed. It is not the case that the sensor performs

[21] This definition intentionally ignores issues of non-determinacy and chaos.

its functions because it has some sort of built-in capability to perform certain functions and that in building the heater we have somehow drawn that capability out of it.

The same is true of all mechanisms, including software. Software is as blind as any other mechanism. A computer controlled by software just goes step-by-step through the instructions written by the programmer: move this value here; add this to that; etc. Those instructions have no sense of the function that the computer is intended to perform. The computer simply does whatever the programmer wrote, no more and no less. There is no implicit functionality waiting to be drawn out of a computer. Perhaps the most difficult hurdle facing beginning programming students is to come to that realization.

Michelangelo is said to have described his work by saying "I saw the angel in the marble and carved until I set him free." Poetic as that may be, there is no sensor functionality waiting to be set free in sensor components, and there is no software functionality waiting to be set free in computer programs. Mechanisms are completely lacking in functional intent—even those that interact with things outside themselves.

But …

Function

Putting materials together in certain ways often *does* produce a certain functionality. That's why software developers write the software they do. That's why engineers design thermostatically controlled heaters they way they do. Software developers and engineers design mechanisms so that the activities that (happen to) occur result in functionalities we want.

Not all functionalities are the product of design; some are serendipitous. Percy Spencer, an engineer at the Raytheon Corporation, was at first surprised to find that the microwave radiation produced by a radar tube could be used for heating. But once he noticed it he was able to use that fact to design the microwave oven.[22]

A similar thing happens in evolution. Mechanisms are created randomly. Some of those mechanisms result in functionalities, and some of those functionalities improve the chances for the entities that possess them to survive and/or reproduce. Those mechanisms are retained in the genome. Often those mechanisms are refined and improved through additional random variation—where improvement means that the revised mechanism performs the function better than the original. Presumably the eye evolved from a light sensitive spot to its current sophistication by such incremental improvement. And sometimes revised designs lead to new functionality. Here are two examples.

When in a medium with a nutrient gradient the bacterium *E. coli* tends to move up the nutrient gradient towards the greater concentration of the nutrient. (Harold 2001) explains how.

> Cells of *E. coli* are propelled by their flagella. … Despite their appearance and name … flagella do not lash; they rotate quite rigidly, not unlike a ship's propeller. … A cell … can rotate [its] flagellum either clockwise or counter-clockwise. … When the flagella turn counter-clockwise [as seen from behind] … the cell [moves] forward in a smooth straight run. … [When] the sense of the rotation is … reversed, the flagellar bundle flies apart and the cell tumbles [randomly]. [Thus *E. coli*] movements consist of short straight runs … punctuated by briefer episodes of random tumbling: each tumble reorients the cell and sets it off in a new direction.

E. coli also includes an internal mechanism that senses whether the concentration of nutrient in its surroundings is greater or less than it was a few milliseconds previously. That sensor is hooked up to the flagella in such a way that if the current concentration is greater, the forward runs are extended; if the concentration is smaller the runs are cut short. So here's the *mechanism*:

(a) Rotating flagella turn either clockwise or counter-clockwise. Clockwise rotation results

---

[22] This is the story according to the Lemelson-MIT Program. http://web.mit.edu/invent/iow/spencer.html.

> [While] touring one of his laboratories [Percy Spencer] stopped momentarily in front of a magnetron, the power tube that drives a radar set. Feeling a sudden and strange sensation, Spencer noticed that the chocolate bar in his pocket had begun to melt. Spencer … did what any good inventor would—he went for some [unpopped] popcorn. … Holding the bag of corn next to the magnetron, Spencer watched as the kernels exploded into puffy white morsels.
>
> From this simple experiment, Spencer and Raytheon developed the microwave oven.

in random tumbling; counter-clockwise rotation results in smooth forward motion.

(b) A sensor measures the concentration of a nutrient and compares it to the concentration a few milliseconds earlier.

(c) The sensor and the rotation mechanism are connected in such a way that counter-clockwise (forward-propelling) rotation continues for a longer period of time if the nutrient concentration is increasing and for a shorter period of time if the concentration is decreasing.

The net effect of that *mechanism* is the *function*: *E. coli* moves in the direction of the higher concentration of the nutrient.

Another example illustrates how a mechanism can even harness external resources to create a functionality. *D. dendriticum*, a liver lancet fluke, spends most of its adult life, including mating, in the liver of grazing animals. After flukes mate the host excretes the fluke eggs. The eggs hatch, and after a stopover in a snail the young flukes wind up in the bodies of ants. Some of the young flukes make their way to the ants' nerve cells. Their presence there (somehow) leads the ants to climb to the tops of blades of grass and wait—as if for a bus—until the grass is eaten by grazing animals—which returns the flukes to their site of reproduction[23].

One might imagine the flukes sitting in an ant "control room" directing the ant to climb the grass and to sacrifice itself for the sake of the fluke. Of course it doesn't happen that way. The ant climbs the grass because of how the fluke (just) happens to interact with the ant's nervous system. What's amazing is not how adept the fluke is at controlling the ant. What's amazing is that evolution managed to create a mechanism as complex as this. As in the case of *E. coli* this example illustrates how a mechanism within an organism (the fluke) results in a functionality that changes the relationship of the entity with its environment.

These examples illustrate how functionality is almost always the result of harnessing an energy flow—even if the energy flow is outside the entity within which the mechanism operates. In the case of *E coli*, the energy flow harnessed was internal to *E. coli*. In the case of the liver fluke the functionality depends to a great extent on actions taken by elements external to the organism. The energy flow harnessed was in the host ant (when it climbs to the top of the blade of grass) and in the grazing animal (when it eats the grass and swallows the fluke).

Purpose

What about purpose? When a functionality produces a benefit for the entity that exercises it—where benefit is defined as a change that makes it more likely that the possessor of the mechanism will persist in the world—we often attribute intentionality to that organism. Why did the eagle swoop down on the rabbit? So that it could feed itself. The ability to swoop down on its prey is a function. The "purpose" of the function is to allow the eagle to eat and survive.

The benefit of the *E. coli* function that propels *E. coli* up a nutrient gradient is that the bacterium will be in position to make use of another bit of functionality, namely its ability to acquire energy from an external source. The benefit of the *D. dendriticum* function that leads its ant host to climb to the top of a blade of grass and wait there until eaten by a grazing animal is that the fluke is returned to a site where it can reproduce.

In both cases one might imagine the organisms that possess these function to be exercising them purposefully—with the intent to realize the available benefit. But functions don't *have* purposes (the purpose is not built into the function) any more than mechanisms *have* functions (the function is not built into the mechanism). When a mechanism *results in* a function that *produces* a benefit, we find it convenient to think of the function—and hence the mechanism—as having a purpose, namely to provide a means to realize the *benefit*.[24]

[23] See the University of Alberta Parasites Lab http://www2.biology.ualberta.ca/parasites/ParPub/text/index/plagi02i.htm.

[24] Of course human beings (often) have more foresight than nature. We create mechanisms (tax codes, for example) that result in functions (tax deductions) that provide benefits (more money) to the people who lobbied for the creation of those mechanism. It makes sense in cases like that to think in terms of intent and purpose.

Categories of entities

In this section I'd like to return to the notion of entities as realized abstractions or persistent patterns or regularities. An obvious question is: how do these patterns persist? This is not a trivial concern. Except at absolute zero, everything is always moving. Without something to keep it in place any pattern will sooner or later lose its orderliness and dissipate.

I have found three answers to how patterns persist. Each answer corresponds to a category of entities.

In my brief answer to the question raised earlier about a common material basis for emergence, I said that energy provides that common basis. I said that there are two categories of naturally emergent entities: static and dynamic. These are two of the categories. I'll discuss them first after which I'll get to the third category, the energy-subsidized entities.

Static entities

The *static entities* are those at an energy equilibrium. These include such entities as atoms, molecules, solar systems, galaxies, etc. Although atoms, molecules, and solar systems are at an energy equilibrium, they are still distinguishable from their surroundings. Each is the embodiment of a pattern in time and space—a pattern that allows one to describe the entity more concisely than a direct enumeration of its parts. In other words, each such entity has less entropy than its surroundings. This entropy difference substantiates the claim that these entities have an objective ontological existence.

What enables the patterns of static entities to persist? These patterns persist because they are in what is sometimes referred to as an energy well. Energy was released when the pattern was created, and energy is required to disrupt the pattern and separate the entity into its components.

Molecules are held together by electro-magnetic forces, atoms by electro-magnetic and atomic forces,and solar systems by gravity. Without an increment of energy each of these will remain in the patterned structure that characterizes it. So the answer to the question "what allows static entities to persist as entities" is that the fundamental forces of physics bind those entities together.

It's worth noting that because the creation of static entities releases energy and energy is required to decompose these entities, static entities have less mass than their components taken separately. Does it sound strange to say, for example, that a molecule of H_2O has less mass than the sum of the masses of the individual hydrogen and oxygen atoms? One way to make this seem less strange is to remember that mass and energy are interchangeable. One produces a molecule of H_2O by "burning" hydrogen. When hydrogen is burned energy, i.e., mass, is released. What remains, the H_2O molecule, has less mass—less by the amount of energy released—than the masses of the elements prior to the reaction. To separate H_2O molecules back into their components requires an addition of energy to replace the energy released when the H_2O molecule was created.

The mass difference between the mass of the H_2O molecule and its components taken separately further substantiates the claim that static entities have an objective ontological existence. They are not just conceptual conveniences.

It may seem that there are relatively few kinds of static entities and that they have a fairly limited repertoire of properties. That is not the case. From semiconductors to superconductors, diamonds to Silly Putty®, superglue to Teflon®, polyester to Gore-Tex®, milk of magnesia to Splenda®, and an impending stream of even more amazing materials to be produced through nanotechnology, static entities are not as uninteresting as they may at first seem.

Dynamic entities

Dynamic entities are stable but are far from an energy equilibrium. These entities are typical of what are frequently referred to as complex systems. Examples are biological organisms and social organizations.

How do these entities maintain their characteristic patterns? The answer differs from one entity type to another. Harold analogizes the metabolic processes in biological cells to an economic system.

> What the economy is to human society, metabolism is to the cell. In short, metabolism is the ensemble of chemical processes by which cells obtain the goods and services required for the continuance, growth, and reproduction. … The metabolic web supplies energy and chemical building

> blocks, and it also carries the information that links the environment, the genome and the molecular machinery into a functional whole.

So how does it all work? After discussing some of the biochemistry details Harold continues as follows.

> Is this pattern of purposeful chemistry laid out in the genome? Not in any identifiable form. … As matters stand, the nucleotide sequence of a gene is not sufficient to let us predict the three-dimensional structure of an enzyme, let alone its kinetic characteristics. The coordinated operation of a metabolic module is quite out of sight.[25]

In other words, we can't yet say in detail how it all comes together. Even if we could, the story would likely involve many accidentally constructed mechanisms whose functionalities turned out to be useful.

> Metabolic networks … are products of evolutionary design. Mutation tinkers with individual genes, but selection judges the performance of organisms.

But what we can say is that running a cell requires energy.

> Enzymic reactions are reversible in principle but unidirectional in practice, and the direction is ultimately imposed by energetics. The second law [of thermodynamics] mandates that all processes in the real world flow down the thermodynamic hill; all entail the degradation of energy, the loss of potential and the production of entropy. Biosynthesis, control and communication all come under this requirement, and can be considered energy-dissipating processes. The dynamic order of the metabolic web, a state far removed from equilibrium, is maintained by the continuous flow of energy from some environmental source to its eventual sink, namely, heat. … The metabolic web illustrates the fundamental principle clearly enunciated by Harold Morowitz [(1968)], that the flows of energy through a system organizes that system. …
>
> The metabolic web … converts energy into organization

The story is the same for all dynamic entities. Like biological organisms, organizational entities—like a tribe, a corporation, a religious community, a bowling league, or a country—often have their own idiosyncratic organizational structures and processes. To understand them requires answers to questions such as the following. Are there formally defined elements such as organizational officers and legislative bodies? What internal processes control the operation of the organization? What are the communication channels? How are decisions made? Etc. The answers may be different from one organization to another, but like a biological organism, each has a structure and a way of operating.

Organizations, again like biological organisms, rely on externally supplied energy to persist. Organizations typically get their energy from their members. It is the members who make the organization work. Without them there would be no organization. In these cases, externally supplied energy may not come directly from raw energy sources. It is provided to the organization through the active participation of its membership.[26]

It's worth pointing out that social organizations, unlike most static entities are not necessarily physically contiguous. A baseball league consists of parts that are spread out across a continent. But organizations must have some means for their components to communicate with each other—either directly or indirectly.[27]

---

[25] Harold continues.

> [It is not the case] that when we know all about the molecular parts we will automatically comprehend how cells are articulated and how they function. … The instructions spelled by the genes are local, not global. But growth and many other cellular operations depend upon energy requiring, directional processes; self-assembly in the cell is directed, in space and in time. … [One should think of a cell] as a partially structured self-organizing system made up of gene-supplied elements. Briefly, the genes specify What; the cell as a whole directs Where and When; and at the end of the day, it is the cell that usually supplies the best answer to the question Why.

[26] In some cases, such as major political entities, organizations accept the responsibility of ensuring that their members have adequate access to energy, i.e. that the members can feed themselves.

[27] The term *stigmergy* was coined to refer to the communication mechanisms that occur indirectly through the environment. Members of an ant colony, for example, leave pheromone trails for each other when they forage for food.

Organizations and organisms are process-based. They consist of components in motion. This is so commonplace an observation that we often don't think about it. But processes and actions are evidence of energy flows. Without energy there is no activity. Because a body in motion has more mass than that same body at rest, dynamic entities possess objectively more mass than their components taken separately.

So again, a mass difference—along with reduced entropy—substantiates the objective ontological existence of dynamic entities. They too are not just conceptual conveniences.

Subsidized entities: you can't break a bit

The third way patterns persist is by being kept in place by external forces. (Dynamic entities use externally supplied energy, but they keep themselves organized.) That's how entities within computational systems persist. Their persistence is guaranteed by the system itself.

Interestingly, not that much maintenance is required. It is not the case that each entity must be individually maintained according to a specially tailored maintenance protocol. If the fundamental building blocks are kept whole, the rest follows.

In computational systems the fundamental building blocks are bits and bit sequences. The bit is the perfect building material. It never deteriorates; it doesn't fracture under pressure; it isn't sensitive to temperature or electromagnetic radiation[28]; it won't rust; it isn't vulnerable to harmonic resonance phenomena; etc. It is because of its absolute reliability—*you can't break a bit*—that the bit can serve as the rock-solid foundation for software-based entities.

How does the bit get those marvelous properties? Electrical engineers design computer systems to ensure that it does. If the computer fails, it's a hardware issue. It's beyond the scope of the entities that occupy the virtual software world. That's why I say that entities within a computer system live in a subsidized world. See (Abbott 2010) for additional discussion.

[28] The hardware that maintains the bit is sensitive to electromagnetic radiation. The bit itself isn't.

How is a Game-of-Life glider like the Ship of Theseus?

A glider is an entity within a computational system. Even though it moves from grid cell to grid cell with each time step, we think of a glider as a single entity. We don't ask whether it is "the same glider" at this time step as it was at some previous time step.

The paradox of the ship of Theseus is to ask essentially the same question. If in the course of maintaining it all the ship's parts have been replaced, is it "the same ship" as it was before the parts were replaced?

The parallel suggests one answer. In both cases, the entity as a pattern persists even though the elements that make up the pattern change from moment to moment.

But what if someone took the materials discarded from the ship and built an identical ship? Would the first (repaired) ship or the second (rebuilt) ship be the "original" ship? The glider example can provide an answer here too. As I mentioned earlier, it's possible to build a so-called glider gun. A glider gun produces an unending stream of gliders. Each successive glider is constructed initially of the same grid cells. But we don't ask whether the second glider, which is made of exactly the same cells as was the first, is really the same as the first glider and whether the first glider, because it now consists of different cells, has become a different glider. Translating that perspective to the example of the ship of Theseus would suggest that a ship built of materials discarded from the original ship is a second ship.

Although the preceding offers one way of looking at the ship of Theseus I prefer a second way. Think of the ship along with the ship repair process as a social entity. That organizational entity includes the physical ship along with the people who do the repairs. As time passes, some people will leave the repair group and other will join. But the group remains the same social organization. It has processes and procedure for ensuring that the ship is kept in good repair.[29]

[29] One might even argue that the process of maintaining the ship helped build community spirit and that by adding that obligation to its social structure Athens strengthened itself as a social organization.

The ship too may find itself completely replaced. But again, the organization of which the ship is a part will persist. It is the group as a social organization that is the persistent entity. The ship is simply one of the material elements of the organization in much the same way that a corporation's headquarters building is one of its material elements. The corporation may move from one headquarters to another—or it may tear down and rebuild its existing headquarters. In either case it remains the same corporation.

What about the ship itself—just the physical ship? My position is that the physical ship changes from one entity to another every time a piece is removed or a new piece is added. If an external force tears part of a static entity away—and the physical ship is a static entity—the entity has changed into a different although similar entity.

That's one of the differences between static and dynamic entities. Static entities are at an energy equilibrium. They exist in an energy well. They don't change unless change is forced upon them, in which case they become something other than what they were before. Dynamic entities typically undergo continual change. They change—but remain the same entity—as a result of their own internal processes, which are powered by energy imported from outside the entity.

The principle of ontological emergence

One might ask which entities will come into existence. My view is that entities come into existence as a result of both physical constraints and historical accidents. We understand how evolution operates. Variations on existing types are created at random. Those that are suited to the environment within which they find themselves persist. The others don't.

It seems to me that a generalization of that process applies to all entities and entity types. They are created more or less at random; the ones that are suited to their environments are the ones that persist. This leads to what I have called (Abbott 2009) the principle of ontological emergence.

> Extant entity types are those whose implementations have materialized and whose environments enable their persistence.

Entities persist if they are successful in the world. As we saw, little is demanded of static entities. They persist if nothing tears them apart. Most familiar chemical compounds, for example, would not persist in an extremely hot and/or turbulent environment. Pour a glass of water into a blast furnace, and the water molecules will come apart. Pour it onto the "surface" of the sun, and even the atoms will come apart.

Dynamic entities must continually supply themselves with energy to persist. Animals must "work for a living." Those that can't find a niche from which they can extract sufficient energy cease to exist.

The "work" that dynamic entities are capable of doing depends on the functionality built into them as instances of particular entity types.

Cows eat grass. That simple statement embeds within it facts about the functionality of the cow entity type. Presumably one could trace at the micro-physical level the processes involved in cows eating and digesting grass. But that would not clarify why the cow entity type succeeds. The cow entity type succeeds as an entity type because its functionality includes the ability of its instances to sustain themselves—as cows—by eating and digesting grass.

For a cow to eat grass requires first of all that there be grass (as another entity type) in the cow's environment. "Cows eat grass" expresses the fact that a cow eats grass with its cow mouth. The microphysical details aren't the issue. The cow opens its cow mouth and closes it over the grass instances. The cow cuts or tears the grass with its cow lips and cow teeth. It uses its cow tongue to move the cut grass around in its cow mouth until it swallows the grass down its cow throat. It then digests the grass in its multiple cow stomachs and intestines and ultimately uses the energy released to maintain itself as a cow entity type instance.

Biological reproduction is even more impressive. One or two parents go through what is sometimes a fairly complex procedure but which results eventually in the creation of a new instance of the same entity type as the parent(s). In many kinds of sexual reproduction an extension of the male becomes enlarged and stiff as a result of being engorged with blood. The male then inserts that extension into a conveniently located and suitably sized orifice of the female and transfers genetic material. The female accepts the genetic material,

combines it with her own genetic material, nurtures the combination within her body (at least in many cases), and then expels it when it is ready to persist on its own in the world.

Again, the micro-physical details could presumably be traced[30], but no microphysical description would adequately convey the process taking place. What matters is that *two instances of an entity type produce a new instance of that type through a process that can only be described in terms of type-level structures and operations*. It is this sort of thing that makes a dynamic entity type successful.

The same is true for every other successful dynamic entity type. A successful dynamic entity type is one that includes functionalities that enable its instances to persist and perhaps even reproduce themselves—*as that type*—in the world in which they find themselves.

Questions from Bedau and Humphreys

In this section I'd like to offer answers to the questions posed by Bedau and Humphreys.

Question 1. How should emergence be defined? A number of leading ideas appear in different definitions of emergence, including irreducibility, unpredictability, conceptual novelty, ontological novelty, and supervenience.

Answer 1. Emergence is best understood as the realization of an abstract data type. The features listed by Bedau and Humphreys are related to emergence, but they don't define it.

Irreducibility seems to me to be an awkward way of characterizing the independence of a specification. It certainly is not the case that emergent phenomena are somehow magically disconnected from their components. As we saw with the glider, one can always explain how a configuration of components produces emergent properties and functionality. Even when there are multiple realizations of an abstraction one can explain how each of the given constructions produces the given functionality. So it isn't as if the emergent properties and functionality do not have a material explanation.

But what typically is the case is that it is not possible to describe the emergent properties and functions in terms of the properties and functions that characterize the components. There is no mapping between the domain of description applicable to the implementing elements and that of the implemented domain. The emergent properties and functions are typically a new and self-contained world that is implemented by—but not derived from or defined in terms of—those of the implementing elements.

This may seem very strange. How is it possible to get from one to the other if there is no mapping? The answer is that although the new abstraction is not mathematically or logically derived from the features of the implementing elements, when put together in certain ways, the implementing elements simply have certain (often new) properties. When carbon atoms are formed into a lattice the resulting diamond is hard even though the notion of hardness does not apply to individual carbon atoms. The same is true of a biological organism. It has properties that do not apply to its chemical components.

Constructions of this sort are standard practice in software. A computer program has properties—e.g., that it computes a particular function—that the instructions and data structures that make it up don't have individually. It is both the components and the way the components are put together that produces the new properties.

That raises the question of the sorts of ways in which things can be put together. When components are put together to form a new entity, certain global constraints are put in place. What are the possible global constraints? That's a very good question and one to which I don't have a good answer.[31]

[30] The microphysical details would not provide information about the amazingly broad range of possible event sequences that may occur. Imagine how one might attempt to describe the sex act in terms of the quarks and electrons involved. It's difficult for me to imagine that what would have to be an equivalent class of activities of fundamental particles could be defined in any meaningful way—other than to say that a male and a female had sex. This is similar to the limitation of supervenience discussed below. Also, see the discussion of a square peg and a round hole in (Abbott 2009).

[31] Computer Science studies the question of the kinds of computations that can be performed within various limits with its categorizations of (a) automata

Here are two more questions for which I have no good answers. (a) What sorts of new properties and functions can one create? (b) What are the implications of the apparent fact that there is no good answer to the preceding question, i.e., that creativity—both natural (through evolution) and human—is in some reasonable sense unlimited?[32]

Unpredictability is simply wrong. Emergent properties and functions are predictable. When one first comes across an instance of a new data type, one may not know how it works. But if a "special science" develops to study the new abstraction and if as is generally the case that special science is successful, the abstraction will be predictable.

Conceptual novelty is usually true. Emergent elements satisfy descriptions that are new and self-contained and that do not derive from the descriptions of their implementations.

I'm not sure what Bedau and Humphreys mean by *ontological novelty*. Emergence and abstract data types have to do with entities and their properties. An abstract data type is a specification that applies to and characterizes instances of that type. Each instance is an ontologically real and distinct entity.

Because emergence is so fundamentally connected to the instantiation of abstractions, entities are central to emergence. It is entities—genereally with new properties and functionalities—that emerge. As discussed above, emergent entities have an objective existence.

The properties of emergent entities *supervene* over the properties of their components. But supervenience is not as simple or useful as one might suppose. Consider a biological organism. At any moment its properties supervene over those of the elements that make it up. But those elements change from moment to moment. Biological organisms are continually shedding parts of themselves and incorporating new elements. So if one were to speak of the "supervenient base" of a biological organism it would consist of all the elements of which the organism is composed over its lifetime. So supervenience holds, but in many cases it is not a very useful conceptual tool for exploring emergence.

The same holds for something as simple as a glider. At any moment a glider supervenes over the cells of which it is made. But the cells that make up a glider change from one time step to the next. So the "supervenient base"—if there is one—of a glider consists of all the cells that ever participate in its makeup. The "supervenient base" of a glider on an otherwise empty Game-of-Life grid is an unboundedly large collection of cells.

Is that useful? In some ways it is. If one were to sabotage just one of those cells before it became part of the glider, the glider would likely fail when it reached that cell. That insight gives us a good way to look at phenomena like poisons and moles (internal spies): control something that will become part of the supervenient base of an entity and one potentially has significant leverage over that entity.

Question 2. What ontological categories of entities can be emergent: properties, substances, processes, phenomena, patterns, laws, or something else?

Answer 2. The elements mentioned (properties, substances, processes, phenomena, patterns, laws) are not mutually disjoint. For example, a biological organism is also a process, and a biological organism has properties and obeys the "laws" that characterize it as an abstraction. My basic answer is as before: emergence is the realization of an abstract data type. Such a realization is an entity, which can have new properties, etc.

Question 3. What is the scope of actual emergent phenomena? This question partly concerns which aspects of the world can be characterized as emergent.

Answer 3. Other than the fundamental elements of physics—if there are any—everything is emergent. See the earlier discussion about regularities.

---

and formal languages and (b) the time and space complexity of algorithms.

[32] It might be worthwhile to formalize the preceding observation. One relatively simple way would be to understand the act of creating a new data type as combining (a) the operation of making tuples (of components) with (b) defining new functions and predicates that take such tuples as arguments—and in the case of functions that return values that may consist of such tuples. At this basic level it seems clear that there is no formal limit to the types one can create and the properties they may have.

Question 4. Is emergence an objective feature of the world, or is it merely in the eye of the beholder?

Answer 4. Emergent entities are objectively ontologically real. See the earlier discussion of entities.

Question 5. Should emergence be viewed as static and synchronic, or as dynamic and diachronic, or are both possible? … In synchronic emergence, the emergent feature is simultaneously present with the basal features from which it emerges. By contrast, in diachronic emergence, the base precedes the emergent phenomenon which develops over time from them.

Answer 5. Emergence can be both synchronic and diachronic. University academic departments, which are (emergent) social organizations, consist of people who come and go and who retain their individual identities even while they are part of the department. The properties of the department are "simultaneously present with the basal features from which it emerges." It is also the case that new departments can be created from people who existed before the creation of the department. In that case "the base precedes the emergent phenomenon which develops over time from them."

Question 6. Does emergence imply or require the existence of new levels of phenomena?

Answer 6. Since emergence is (in my view) the realization of an abstraction the elements that realize the abstraction are in a micro to macro relationship to the realized abstraction as a whole.

But that doesn't mean that nature is a tiered hierarchy. One of my favorite examples is the gecko. Geckos are macro level entities that rely directly (Kellar 2002) on the quantum-level Van der Waals force between hairs on their feet to the vertical surfaces to which they can adhere. Yet there are many intermediate "levels"—such as atomic physics, chemistry, biochemistry, etc.—between quantum forces and geckos.

Question 7. This question bundles together a number of issues. I'll split them apart.

Question 7a. In what ways are emergent phenomena autonomous from their emergent bases? … A number of different kinds of autonomy have been discussed in the literature, including the ideas that emergent phenomena are irreducible to their bases, inexplicable from them, unpredictable from them, supervenient on them, and multiply realizable in them.

Answer 7a. This sub-question itself bundles together a number of sub-sub-questions. Most have already been addressed, but I'll respond briefly to each.

In what ways are emergent phenomena autonomous from their emergent bases? Emergent phenomena are autonomous in that they are characterized by independent, self-contained specifications.

Are emergent phenomena irreducible to their bases? Emergent phenomena are generally not reducible if reducibility is taken to require a kind-to-kind mapping. That's the position Fodor (1974) took three and a half decades ago. I think he was right. Emergent phenomena are characterized by autonomous specifications, which cannot in general be mapped directly to the kinds, properties, and functionalities available in their bases.

Are emergent phenomena inexplicable from their bases? No. Emergent phenomena may be conceptually novel with respect to the conceptual apparatus needed to describe their bases. But it is always possible to explain how the emergent phenomena have the properties and functions they do by examining how they are realized. As we saw with the glider, the philosophical approach known as reductive explanation will typically do the job.

Are emergent phenomena unpredictable from their bases? No. See my reply to *predictability* in question 1.

Are emergent phenomena supervenient on their bases? Yes, but see my comments on supervenience in my reply to question 1.

Are emergent phenomena multiply realizable in their bases? Frequently, but multiple realization is not as important is it is often made out to be. Multiple realization is a consequence of emergence rather than a criterion required for it.

When Putnam (1975) described autonomous phenomena from a functionalist perspective he emphasized the importance of functional isomorphism between two physical entities.

> [T]wo systems can have quite different constitutions and be functionally isomorphic.

> For example, a computer made of electrical components can be isomorphic to one made of cogs and wheels.

It was an important insight that radically different entities can be functionally isomorphic. This insight began to clarify what is meant when one says that emergent phenomena (or their properties or their functions) are autonomous of the elements (and the properties and functions of those elements) of which they are composed.

Unfortunately (in my view) Putnam's insight has become a primary focus in establishing the autonomy of a functional specification. If, the argument goes, one can produce two or more elements that have (as Putnam says) "quite different constitutions" but that yet are "functionally isomorphic," then presumably that functionality is in some reasonable sense independent of or autonomous from the elements' constitutions.

Although this may be true, I think it misses the point. As Putman points out "a computer made of electrical components can be isomorphic to one made of cogs and wheels." But it is not the isomorphism between the different computers that establishes that the functionality of a computer is autonomous of its implementation. The autonomy derives from the fact that the functionality of a computer can be specified independently of its implementation.

What is the functionality of a computer? It certainly doesn't consist of all the properties and functions possessed by a computing device made of cogs and wheels. Nor is it defined by all the properties and functions possessed by a computing device made of electrical components. Many of these properties will have something to do with cogs and wheels or with electrical components. To establish the isomorphism one strips away the irrelevant properties. The abstraction that remains—to which both realizations can be mapped and which can map to both realizations—is what characterizes what one means by the functionality of a computer.

But having developed that abstraction, it becomes clear that what one means by the functionality of a computer can be specified abstractly even if the only physical example were one made of cogs and wheels—or even if there were *no* physical examples at all.

Multiple realizability has nothing to do with the independence and autonomy of such an abstract specification. On the contrary, it is the autonomy of the specification that enables multiple realization. Once one has abstracted out the functionality one cares about, one sees that one can create many quite different devices that produce that same functionality.

Because it is the independence of a specification that establishes autonomy I'm puzzled by Fodor's objection to Game-of-Life examples. Fodor (1997) refers to an article of Dennett (1991) that notes that it is possible to use the Game of Life to implement a Turing machine. Fodor argues[33] that

> when macroregularities are logical or mathematical constructions out of microregularities, as in the "Game of Life" described in Dennett (1991), laws about the latter imply laws about the former. Pace Dennett, such cases do not illuminate (what functionalists take to be) the metaphysical situation in the special sciences.

The problem I have with Fodor's objection is that the theory of computability—which defines the "laws" governing Turing machines—is not in any useful sense a logical or mathematical construction derived from the rules of the Game of Life.[34]

---

[33] paraphrased from a footnote in the extract quoted in full in Appendix A.

[34] One might argue that since the Game-of-Life rules do not conflict with the theory of computability any derivation of the theory of computability would still work if the starting point of that derivation also included the Game of Life as part of the basis of the derivation—e.g., if Turing had invented a broader class of automata that included both the Game of Life and the Turing machine but then focused exclusively on the Turing machine.

That argument would imply that the theory of computability *is* derivable from the Game-of-Life rules. Although that argument holds, that's presumably not what Fodor had in mind when he suggested that the theory of computability is a logical or mathematical construction based on the rules of the Game of Life.

The situation to which Fodor is referring is that in which a Turing machine is implemented within—or realized by—the Game of Life. But that seems to me to be parallel to Putnam's argument that the functionality of a computer can be realized by a number of different kinds of devices. In both cases one has an independently specified functionality realized by

The theory of computability is paradigmatically autonomous from the laws governing the Game of Life. In fact Turing developed the theory of computability long before Conway invented the Game of Life.

Question 7b. Is … autonomy … merely epistemological or [does] it [have] ontological consequences?

Answer 7b. Although emergent entities are ontologically real their reality is not a consequence of their autonomous specification. Emergent entities are ontologically real because of the physics of their construction.

Question 7c. [Does] emergence necessarily [involve] novel causal powers, especially powers that produce ''downward causation,'' in which emergent phenomena have novel effects on their own emergence base.

Answer 7c. As far as I am concerned there is no such thing as downward causation. The only forces of nature—and presumably the only source of causality in the sense intended here—are the fundamental forces of physics. But there is a very important qualification. The existence of an emergent abstraction can result in what—in (Abbott 2006 and 2009)—I called downward entailment.

Downward entailment refers to an important aspect of the relationship between a realized abstraction and the elements that realize it. When an abstraction is being realized, the elements that realize the abstraction participate in whatever happens to the realized entity. Suppose I throw a baseball. One of the consequences of my throwing the baseball is that it moves from here to there. As long as the baseball is being realized by its components, those components will also move from here to there.

This is not downward causation. One could (presumably) explain the movement of each component of the baseball at the level of quantum physics. There are no new forces that apply to baseballs and other macro elements. Furthermore, if the elements that made up the baseball stopped making it up—suppose the baseball lost its cover and its innards unwound in mid-flight—the abstraction would cease to be realized, and the elements that had previously made up the realized abstraction would no longer participate in its (former) motion.

As another example consider the implications of the fact that it is possible to implement Turing machines in the Game of Life. Since it is undecidable whether a Turing machine will halt, it is undecidable whether an arbitrary Game-of-Life configuration will reach a stable state.

This is a direct example of downward entailment. A fact about Turing machines has implications for the Game of Life itself. As long as the Game of Life is implementing a Turing machine, facts about the Turing machine abstraction will apply to those elements of the Game of Life that participate in its implementation.

This illustrates reduction in mathematics and computer science. When something can be reduced to something else, conclusions about the latter apply to the former. In this case the halting (or stability) problem for the Game of Life has been reduced (upward!) to the halting problem for Turing machines—which then entails (downward) facts about the Game of Life. The reduction is done by implementing Turing machines within the framework of the Game of Life.

Conclusions: analysis vs. construction

As I mentioned earlier, one of the corollaries of looking at emergence the way I am proposing is that it highlights nature's creativity. Putting things together in new ways can produce new properties and functions.

The emergence-as-creativity perspective illustrates one of the differences between computer science (along with the creative arts, much of engineering, and parts of mathematics) and most analytical disciplines. Computer science is inherently constructive/creative. Software developers (and many engineers, mathematicians, and creative artists) imagine new things and figure out how to put existing things together to realize them. Practitioners in analytical fields generally

a construction that implements the functionality. What matters in such a situation is (a) that the implementing mechanism not be in conflict with the implemented functionality and more importantly (b) that the implementing mechanism be powerful enough to implement the realized functionality. That's the situation with both a cogs-and-wheels computer and a Game-of-Life Turing machine.

attempt to determine how existing things came to be the way they are.

REFERENCES

Abbott, Russ (2006) "Emergence explained," *Complexity*, Sep/Oct, 2006, (12, 1) 13-26. Preprint: http://cs.calstatela.edu/wiki/images/9/95/Emergence_Explained-_Abstractions.pdf

Abbott, Russ (2008) "If a tree casts a shadow is it telling the time?" *International Journal of Unconventional Computing*, 4/3, pp 195-222. Preprint: http://sites.google.com/site/russabbott/Ifatreecastsashadowisittellingthetime.pdf

Abbott, Russ (2009) "The reductionist blind spot," *Complexity,* to appear. Also presented at the *North American Conference on Computers and Philosophy*, July 2008, Bloomington, Indiana, 2008. Preprint: http://cs.calstatela.edu/wiki/images/c/ce/The_reductionist_blind_spot.pdf.

Abbott, Russ (2010) "Bits don't have error bars: upward conceptualization and downward approximation," in van de Poel, Ibo and Goldberg, David E. (Eds.) *Philosophy and Engineering: An Emerging Agenda*, Springer. Presented originally at the *Workshop on Philosophy and Engineering*, Delft, The Netherlands, 2007,

Bedau, Mark and Paul Humphreys (2008) *Emergence*, MIT Press. Introduction available: http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11341.

Chaitin, G. (2009) "Leibniz, Complexity and Incompleteness," *APA Newsletter on Philosophy and Computers*, Vol. 9, No. 1 (Fall 2009), pp. 7-10. Retrieved from: http://cswww.essex.ac.uk/staff/turnr/Mypapers/v09n1_Computers.pdf.

Chapman, Jason W., Rebecca L. Nesbit, Laura E. Burgin, Don R. Reynolds, Alan D. Smith, Douglas R. Middleton, and Jane K. Hill (2010) "Flight Orientation Behaviors Promote Optimal Migration Trajectories in High-flying Insects," *Science*, American Association for the Advancement of Science, Vol. 327. no. 5966, pp. 682 – 685, 5 February 2010.

Dennett, Dan (1991) "Real Patterns," *Journal of Philosophy*, 87, 27-51.

Fodor, Jerry A. (1974): "Special sciences and the disunity of science as a working hypothesis", *Synthese*, 28, pp 77-115.

Fodor, Jerry A. (1997) "Special Sciences; Still Autonomous after All These Years," *Philosophical Perspectives, 11, Mind, Causation, and World,* pp 149-163.

Holland, J. (1997) *Emergence: From Chaos to Order*, Addison-Wesley.

Kellar Autumn; Metin Sitti; Yiching A. Liang; Anne M. Peattie; Wendy R. Hansen; Simon Sponberg; Thomas W. Kenny; Ronald Fearing; Jacob N. Israelachvili; Robert J. Full (2002) "Evidence for van der Waals adhesion in gecko setae." *Proceedings of the National Academy of Sciences of the USA*, *99*, 12252-12256.

Harold, Franklyn M. (2001) *The Way of the Cell: Molecules, Organisms, and the Order of Life*, Oxford University Press.

Margulis, Lynn and Dorion Sagan (1986) *Microcosmos: Four Billion Years of Evolution from Our Microbial Ancestors.* Summit Books.

Morowitz, Harold (1968) *Energy Flow in Biology: Biological Organization as a Problem in Thermal Physics*, (Reprinted 1979 by OxBow Press, Woodbridge, CT.)

O'Connor, Timothy and Hong Yu Wong (2009) "Emergent Properties", *The Stanford Encyclopedia of Philosophy* (Spring 2009 Edition), Edward N. Zalta (ed.), http://plato.stanford.edu/archives/spr2009/entries/properties-emergent/.

Pearl, Judea (2000) *Causality: Models, Reasoning and Interference*, Cambridge University Press.

Putnam, Hilary (1975) "Philosophy and our Mental Life." In *Mind, Language, and Reality*, 291-303. Cambridge University Press.

Quine, W. V. O. (1953) "On What There Is", as reprinted in *From a Logical Point of View*, New York: Harper & Row.

Robert, Henry M. III, et. al. (2000) *Robert's Rules of Order (10th Edition)*, Da Capo Press.

Shalizi, C. (1998) "Review of Emergence from Chaos to Order," *The Bactra Review*, http://cscs.umich.edu/~crshalizi/reviews/holland-on-emergence/

Smith, B. (2003), "Ontology," in Floridi *Blackwell Guide to the Philosophy of Computing and Information*, Oxford: Blackwell, 2003, 155–166.

Turing, A. M. (1936) "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society* 1937 s2-42(1):230-265.

Weisstein, Eric W. "Peano's Axioms." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/PeanosAxioms.html

Woodward, James (2003) *Making Things Happen*, Oxford University Press.

Table 1. Table of entity types (from Abbott 2009).

		Naturally occurring	***Human designed***
Energy status	***Static.*** At an energy equilibrium; in an "energy well." ——— *Supervenience is useful.*	Atoms, molecules, solar systems, … ——— *Homeostatic mechanisms:* lowest energy state.	Tables, boats, houses, cars, ships, … ——— *Homeostatic mechanisms:* few; generally dependent on "maintenance" processes,
	Dynamic. "Far from" an energy equilibrium. Must import energy (and usually other resources) to persist. ——— *Supervenience is not useful.*	Hurricanes(!), biological organisms, biological groups such as ant colonies, … ——— *Homeostatic mechanisms:* specialized for individual cases.	Social groups such as governments, corporations, clubs, the ship of Theseus(!), … ——— *Homeostatic mechanisms:* specialized for individual cases, ranging from force to incentives.
	Subsidized. Energy is not relevant since it is provided "for free" within an entity "incubator." ——— *Supervenience is not useful.*	Ideas, concepts, "memes," … . The elements of a conceptual system. (This paper is not about consciousness. This category just fits here.) ——— *Homeostatic mechanisms:* We don't understand how concept formation works.	The "first class" values—such as objects, classes, class instances, etc.—within a computational system. ——— *Homeostatic mechanisms:* generally not required since no natural degradation.

Appendix. Extract from Fodor 1997

The very *existence* of the special sciences testifies to reliable macrolevel regularities that are realized by mechanisms whose physical substance is quite typically heterogeneous. Does anybody really doubt that mountains are made of all sorts of stuff? Does anybody really think that, since they are, generalizations about mountains-as-such won't continue to serve geology in good stead? Damn near everything we know about the world suggests that unimaginably complicated to-ings and fro-ings of bits and pieces at the extreme *micro*level manage somehow to converge on stable *macrolevel* properties.

On the other hand, the 'somehow' really is entirely mysterious, and my guess is that that is what is bugging Kim. He just doesn't see why there should be (how could there be) macrolevel regularities *at all* in a world where, by common consent, macrolevel stabilities have to supervene on a buzzing, blooming confusion of microlevel interactions. Or rather, he doesn't see why there should be (how could there be) unless, at a minimum, macrolevel kinds are *homogeneous* in respect of their microlevel constitution.[35] Which, however, functionalists in psychology, biology, geology and elsewhere, keep claiming that they typically aren't.

So, then, *why is there anything except physics*? That, I think, is what is *really* bugging Kim.[36] Well, I admit that I don't know why. I don't even know how to *think about* why. I expect to figure out why there is anything except physics the day before I figure out why there is anything at all, another (and, presumably, related) metaphysical conundrum that I find perplexing.

35 [This is Fodor's footnote 5.] Note that what's mysterious isn't macrostructure per se; it's irreducible (autonomous) macrostructure. When macrokinds are metaphysically identical to microkinds, laws about the latter imply laws about the former; likewise when macroregularities are logical or mathematical construction out of microregularities, as in the "Game of Life" described in Dennett (1991). Pace Dennett, such cases do not illuminate (what functionalists take to be) the metaphysical situation in the special sciences. To repeat: autonomy implies 'real' (vis projectible) patterns *without reduction*.

36 [This is Fodor's footnote 6]. To be sure Kim is also bugged about problems of causal and explanatory overdetermination. [The rest of the footnote is not included here.]