

Emergence, reduction, and concept grounding in philosophy and computer science

Russ Abbott

California State University, Los Angeles
Los Angeles, CA 90032 USA

russ.abbott@gmail.com

Keywords

computer science, concept grounding, emergence, model theory, philosophy, reduction, reductive explanation, symbol grounding, thought externalization.

Abstract

Software generally serves as a model—in the model theory sense—for abstractions, which because they are specified independently of their implementation may be understood as autonomous—and hence emergent—phenomena. By providing reductive explanations software takes the mystery out of emergence and provides a paradigmatic example of how emergence works. This approach to emergence is commonplace in computer science but relatively rare in philosophy, which tends to look for explanations of emergence that follow a more traditional reductive approach.

Since software can serve as a model for a wide variety of abstractions, it provides a concept grounding mechanism for computer scientists. No comparable grounding mechanism is available in other disciplines.

These two features of software may explain the difference in how emergence is understood in computer science and philosophy.

The grounding problem

This introductory section introduces grounding as a problem that applies to both symbols and concepts. It introduces (informal) model theory as a framework within which to discuss these problems. And it suggests that software can enable computer scientists¹ to externalize—and hence ground—our concepts.

The symbol grounding problem

The symbol grounding problem (Harnad 1990) asks how a symbol manipulating device (like a computer) can connect the symbols it manipulates with their referents.

A decade before Harnad formulated the symbol grounding problem Searle (1980) argued, without using the term, that symbol grounding for computers was not possible.

What [a computer] does is manipulate formal symbols. The fact that the programmer and the interpreter of the computer output use the symbols to stand for objects in the world is totally beyond the scope of the computer. The computer ... has a syntax but no semantics.

This argument has led to work in robotics and other physically embodied systems that are embedded (sometimes called situated) in physically environments.

Model Theory

Model theory (e.g., Hodges 2008) studies connections between a language (i.e., a set of statements, propositions, etc.) and a model, where the model consists of a collection of objects along with relationships among those objects.

Given linkages from a language to a model the model defines a semantics for the language. The model is what the language is talking about. Facts about the model provide meanings for statements in the language, and they enable one to say which statements are true (of the model).

As a simple example, consider the sentence “All men are mortal.” Consider the model consisting of the universe of motion pictures. If one connects “men” to (all of the elements in the domain of) motion pictures and takes “mortal” to mean has finite running time then, “All men are mortal” is true in this model.

Model theory has developed into a powerful and sophisticated branch of mathematics. This paper

¹ When I refer to computer scientists, I include myself.

makes use only of the intuitive conceptual framework just described, the mapping of symbolic expressions to things and their relationships.

The goal of symbol grounding is to create the sorts of mapping from symbols to objects studied by model theory. Model theory doesn't deal with the question of how a mapping from symbols in a computer to things in any real-world context might actually be constructed. That is, it doesn't solve the symbol grounding problem. But model theory does give us a straightforward way to understand what we mean by symbol grounding.

The concept grounding problem

This paper is not about a theory of mind. It does not ask how minds are able to form, manipulate, and understand concepts—or even what it means for a mind to form, manipulate, and understand concepts. But based on my personal (subjective) experience, I feel comfortable saying that my mind does form, manipulate, and understand concepts. I also believe that yours does too and that it does it in more or less the same way as mine.

So for this paper I am taking as given (a) that minds form, manipulate, and understand concepts and (b) that we all understand at an intuitive and personal level what it means for a mind to form, manipulate, and understand concepts.

The concept grounding problem may be defined analogously to symbol grounding as the problem of how we as concept manipulating beings connect concepts to anything outside our minds.

Mental models and mental states

In his encyclopedia article on intentionality, Byrne (2006) offers the notion of *the representational content* of a mental state—the collection of propositions for which that mental state is a model²—as a useful way to talk about intentionality.

Byrne doesn't specifically say what he means by a mental state. But the basic idea, at least as I want to understand it, is that a mental state has to do with subjective experience.

Subjective experience covers a lot of ground. For this article, I'd like to limit the discussion to sub-

jective experience involving concepts. Since the representational content of a mental state is the collection of propositions for which that state is a model, what we informally call a mental model may also be understood more formally as a model of its representational content.

To be clear, my sense is that mental models generally *precede* the propositions for which they serve as a model. We often get an idea in our minds and then struggle to find the right words to express and explain it.

Going in the other direction, from propositions to mental states, is sometimes even harder. When I talk to you, I am asking you to take my words and form a mental model for those words. Reading—with comprehension—is paradigmatically the task of going from words to mental models.

With this as background the concept grounding problem may be understood as two steps.

- i) Construct the representational content of (i.e. propositions that reflect) our mental models, i.e., put our thoughts into words, and
- ii) Ground the symbols in those propositions, i.e., connect those words to things outside our minds. (This second step amounts to the symbol grounding problem for those propositions.)

In taking this position I am not claiming that all grounding of subjective experience is conceptual grounding. It seems to me that we frequently—probably more often than not—interact with the world without constructing representational content propositions as intermediaries. How we ground our mental models (with or without making use of representational content propositions) is, of course, one of the great open questions.

Abstract data types and models

An abstract data type is a specification of (a) abstract objects, (b) relationships among those abstract objects, and (c) operations that may be applied to them. A software implementation of an abstract data type is therefore a model of that specification.

As an example, Microsoft Word implements such abstractions as paragraphs, words, fonts, pages, documents, etc. The statements about paragraphs, words, fonts, pages, etc. that are true in the Microsoft Word model of these abstractions

² The use of model theory terminology in this context is my formulation, not Byrne's.

are (tautologously) those that describe how Microsoft Word implements these abstractions.

Software is almost always the implementation of some abstraction. The propositions that describe the abstraction are sometimes referred to as a specification of the software. Alternatively, they are sometimes referred to as requirements for the software, and they are sometimes referred to as user documentation. Such descriptions are (appropriately) almost always about the user-level abstractions that the software implements, i.e., the abstractions for which the software is a model.

Is it reasonable to think of software as a model?

One might object to describing software as a model on the grounds that model elements tend not to be composed of other, lower level components. When we say that the natural numbers are a model of Peano's axioms, we don't think of each natural number as being built from simpler stuff.

Yet we know that computer programs are composed of statements in a programming language—and that when executed these statements consist of sequences of machine language instruction, which in turn consist of the operation of logic gates, etc.

The fact that software is composed of simpler elements is not relevant when we consider it in its role as a model. One might think of software (generically) as “modeling clay” that can be molded by programmers into models for a wide variety of symbolic abstractions.

In this sense software is something of a universal symbolic modeling material—just as general purpose computers are general purpose and universal Turing machines are universal. But once a programmer creates a software model of an abstraction, the fact that the modeling “clay” itself is composed of “clay molecules” is not relevant. All that matters is whether statements that describe the abstraction are true in the model.³

³ These considerations apply beyond software. They apply whenever a model is something other than a mathematical structure. For any real-world model the elements of the model are objects in the world. Unless they are elementary particles they are composed of simpler components. But as far as using those elements as a model, it is only the model level

Software and concept grounding

Software manipulates symbols. (See Abbott 2007.) As far as I know, it is the only general purpose non-biological artifact that does. The ability to write software enables software developers to construct symbol-based models that operate outside the human mind.

Because software offers this capability, computer scientists are able to ground our mental models (see Abbott 2008a). We are able to perform the two concept-grounding steps sketched above. (i) Like everyone else, we can formulate our concepts by constructing symbolic expressions to describe them. (ii) Unlike most others we can build software models that ground those symbolic expressions.

Not only are we able to ground our concepts, the concept-grounding steps are an everyday part of our lives. A typical computer science activity starts with a concept to be implemented.⁴ The software developer then proceeds: (a) to formulate a symbolic expression (often informal) for that concept and (b) to construct a software model of that symbolic expression.

Because this is so much a part of our everyday lives, computer scientists take model construction as second nature. Furthermore, once a concept is externalized and grounded our understanding of what the concept means can often be transferred from our internal conceptual models to how the externalized model of the concept actually functions.

As far as I know, conceptual grounding is not available to other disciplines such as philosophy. Philosophers can externalize concepts as words, but there is no device other than a human being that is able to manipulate words as symbols. The words by themselves don't create a model.

Philosophers can also formalize concepts as predicate calculus. But predicate calculus is on the propositional side of the proposition/model divide and does not serve to ground concepts. In

that matter, not how elements at that level are built of simpler real-world stuff.

⁴ That concept may have been created by the software developer herself, or it may have arisen from an external source. In either case, the concept is the starting point.

other words, there is no way for philosophers to get out of their minds.

An example

To illustrate the difference in grounding between computer science and philosophy consider the concept of a paragraph. The American Heritage Dictionary defines⁵ “paragraph” as

A distinct division of written or printed matter that begins on a new, usually indented line, consists of one or more sentences, and typically deals with a single thought or topic or quotes one speaker's continuous words.

If a philosopher were to elaborate that definition, he would presumably note, among other things, that the term apparently applies only to written text, that it is one of possibly a number of ways of dividing text into components, and that it consists of one or more subcomponent elements called sentences. He would also note that the notion of a paragraph seems to embody (a) semantics (that it deals with a single thought or topic), (b) syntax (as noted, that it is composed of sentences), and (c) appearance (that it begins on a new line, which is usually indented). Further analysis would ensue.

A computer scientist would limit herself to characteristics that could be implemented in software. In Microsoft Word, for example, a paragraph is what one gets when one triple clicks the text. A paragraph is also the unit of composition to which a style (font, line justification, spacing before and after, margins, etc.) is applied.

The computer science definition is more superficial than the philosophical definition. But it is grounded and concrete. In the end, for a computer scientist, a paragraph is what a particular piece of software treats as a paragraph.

This is not to say that a computer scientist must give up her intuitive idea of a paragraph. Considering how primitive our ability to write software that processes language is, the software version of a paragraph will be an inadequate replacement for our intuition—although that may not always be true. In such a case one compares the software implementation with intuition. If one is not happy with the way the software models paragraphs, the software can be changed. But ulti-

mately, for a computer scientist the concept of a paragraph is externalized, modeled, and grounded as software, and its meaning is transferred from an internal concept to whatever the software does.

The philosophical definition of a paragraph is far richer. But it is ungrounded. What is a “single thought,” for example? In the end philosophical analysis may help one develop a better understanding of what one wants to think of as a paragraph—and hence how software should model them. But it’s hard to imagine how philosophical analysis can pin down the notion of a paragraph in a way that doesn’t depend on other ungrounded concepts.

Why do philosophers find emergence mysterious?

The preceding discussion grew out of my attempt to formulate and then answer the question that heads this section. The question itself arose from my experience working with emergence over the past couple of years.

Emergence is the notion that there can be (higher level) phenomena that are autonomous from the (lower level) phenomena from which they arise.

I have written (2006, 2007, 2008a, 2008b, 2009) that a computer science approach to emergence clarifies and resolves the fundamental issues. One of my purposes in writing for the APA Newsletter is to find out what the philosophical community has to say about this approach.

The philosophical approach to emergence

For the most part, philosophers tend to see emergence as mysterious. As recently as April 2008 in the introduction to their edited collection of articles about emergence, Bedau and Humphreys asserted that “the very idea of emergence seems opaque, and perhaps even incoherent.”

Yet it isn’t as if the notion that higher level phenomena may be autonomous from a lower level base is not well known in the philosophical literature. For more than three decades, functionalist philosophers (e.g., Fodor 1974 and 1997) have argued for the autonomy of the special sciences—any science other than physics. In a widely quoted passage, Fodor puts it this way.

The very *existence* of the special sciences testifies to reliable macro-level regularities ... Damn near every-

⁵ <http://www.bartleby.com/61/87/P0058700.html>.

thing 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. ...

But, Fodor continues (slightly paraphrased),

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. ...

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.

It’s not just Fodor, Bedau, and Humphreys who are puzzled by emergence. From Putnam (1975) to Kim (2006) emergence has provoked years of philosophical bewilderment.

In 1975 Putnam described autonomous emergent properties from a functionalist perspective.

Mentality is a real and autonomous feature of our world. ... [Its autonomy] has nothing to do with ... that all too old question about matter or soul-stuff. We could be made of Swiss cheese and it wouldn’t matter. [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.

Unfortunately (in my view), Putnam’s paper drifts away from the important issue, the autonomy of higher level functionality. For one thing, it focuses on mental states—an area that is so scientifically and philosophically unsettled that the uncertainties surrounding it inevitably contaminate any discussion in which it appears.

Secondly, Putnam discusses the similarities and differences between human beings and Turing machines, which seems to me to be a major distraction.

Finally, Putnam emphasizes functional isomorphism between two physical entities. This latter concern has blown up into what seems to me an unnecessary obsession with multiple realizability.

It was clear from the start—Putnam says so himself—that a Turing machine is functionally autonomous of its implementation. Turing machines are simply specified independently—and hence are autonomous of any implementation. Multiple realizability has nothing to do with it. It is the au-

tonomy of the specification that enables multiple realization, not multiple realization that establishes the autonomy of the specification.

A computer science approach to emergence

Although computer science doesn’t tend to use the term “autonomous” in this context, for the purposes of this paper higher level abstractions may be considered autonomous if they can be described or specified independently of their implementation. As noted above, this is the fundamental idea behind the computer science notion of an abstract data type.

Virtually all software creates emergent autonomous phenomena. As already discussed, Microsoft Word implements such (emergent) autonomously specified abstractions as paragraphs, words, fonts, pages, documents, etc. These concepts and the rules that govern them are autonomous from the lower level phenomena (statements in a programming language, machine instructions, logic gates, voltage dynamics, etc.) upon which they are built. None of the underlying levels has anything to do with documents, paragraphs, words, characters, or fonts.

Yet there is no mystery about how such autonomous higher level abstractions come about: software creates them. In fact, a reasonable characterization of computer science is that it is the art and science of using lower level abstractions to design and build higher level (emergent) abstractions. (See Abbott 2008b, 2009.)

Similar processes produce emergence in the world of more material phenomena. We know how to take two protons and two electrons and create a hydrogen molecule (H₂)—which has properties far different from isolated protons and electrons. We know how to put a hydrogen molecule together with an oxygen atom to form a water molecule—which, when combined with other water molecules at the right temperature to create a liquid, has properties far different from hydrogen and oxygen separately.

We know that a sodium atom and a chlorine atom combine to form something that tastes⁶

⁶ Although the taste of salt is a subjective experience, the fact that NaCl binds chemically to the salt receptors in the tongue is not.

salty, although neither does individually. We know: that a team of 11 people can perform intricate football plays; that colonies of ants, bees, or termites can function as colonies; that packs of wolves can function as packs; and that a group of four musicians can function as a string quartet. And we know that the cells in our bodies can function as a human being.

We may be awed by some of these feats. But we don't consider them mysterious or incoherent. So why the continuing mystery about emergence within the philosophical community?

Reduction vs. modeling

In considering this puzzle I started to wonder whether it isn't a matter of the different ways in which philosophers and computer scientists look at emergence and related concepts. Once I asked that question it struck me that there were a number of concepts that computer science and philosophy approach somewhat differently. In this note, I consider reduction (philosophy) vs. model construction (computer science). Other contrasts that will have to wait for another time are

- supervenience (philosophy) vs. functional dependency (computer science)
- causality (philosophy) vs. execution (computer science), and
- kind (philosophy) vs. (type) computer science.

This section explores the differences between traditional reduction—in which a reducing domain is mapped onto a domain to be reduced—and modeling (also now known philosophically as reductive explanation)—in which a domain to be reduced is modeled within a reducing domain.

Reduction

Traditional philosophic reduction is taken to mean that the domain (problem, theory, computation) being reduced is in some sense a direct consequence of the reducing domain. The reduc-

But it's better to think of this the other way around. Biological organisms evolved chemical sites to which salt would bind because having those sites was advantageous for survival. In evolving these receptors we developed a chemical model for the environmentally implicit specification: something to which NaCl would bind.

tions of (a) thermodynamics to statistical mechanics⁷ and (b) Newtonian mechanics to relativistic mechanics are frequently offered as examples.

This approach to reduction was originally formulated by Nagel (1961). His bridge laws are intended to map the reducing domain to the domain to be reduced, thereby showing that the domain to be reduced is little more than a repackaging of features of the reducing domain.

To take a simple mathematical example: how many distinct subsets can one create from a collection of n elements? (Include as possibilities the empty set and the entire original collection.)

This problem can be reduced to that of determining how many distinct strings of n bits are there? But this second problem is trivial. There are 2^n distinct strings of n bits, namely the numbers from 0 to $2^n - 1$. So if the reduction works, there are 2^n possible subsets of a set of n elements.

How does one do the reduction? One can map one-to-one (each distinct string represents a distinct subset, and each subset is represented by a distinct string) from strings of n bits to subsets of a collection of n elements. For a given bit string, think of the i^{th} bit as representing whether or not the i^{th} member of the collection is in the subset represented by that bit string. The string of n 0's represents the empty subset; the string of n 1's represents the entire collection.

Furthermore, once one has constructed this reduction, one can map the bit string operations bitwise AND and OR to set union and intersection.

Reductions of this sort might be described as projections from the reducing domain to the reduced.

Although the preceding may be a bit of an oversimplification of traditional reduction, you can see the problem. If reduction requires a relatively simple mapping from the reducing domain to the domain to be reduced, the domain to be reduced must be no more complex than the reducing domain. It must have no more types (kinds) and no more relationships. Otherwise it would not be

⁷ The claimed reduction of thermodynamics to statistical mechanics is no longer taken as a slam dunk. See (Howard 2007), for example.

possible to create a simple map onto it from the reducing domain.

A consequence is that there is no room for autonomous phenomena in the domain to be reduced. Requiring a straightforward mapping from the reducing domain to the domain to be reduced precludes any autonomy in the domain to be reduced.

Perhaps it is a perspective of this sort that makes the possibility of the emergence of autonomous phenomena seem incoherent.

Models and reductive explanation

Computer science works with a form of reduction in which the domain to be reduced may be more complex than the reducing domain. In this form of reduction one builds a model of the domain to be reduced within the reducing domain.

A standard example is the reduction of Turing computability to Game-of-Life computability. It is possible to model Turing machines within a Game-of-Life framework. In doing so, one does not map Game-of-Life features (such as grid cells) to Turing machine features (such as tape cells). Nor does one map Game-of-Life operations (such as cells turning on or off) to Turing machine operations (such as a cell being written on by the Turing machine read-write head). Instead one builds a model of a Turing machine by using elements available within the Game of Life.

The term ‘model’ in the preceding paragraph can be taken formally. The domain to be reduced may be understood as a specification—in this case a specification of a Turing machine. To accomplish the reduction one builds within the reducing domain of an operational model of that specification.

Reductions of this sort are really constructive: a model of the domain to be reduced is implemented (constructively) using mechanisms available within the reducing domain. This is quite different from simply projecting the reducing domain forward through bridge laws to generate properties of the domain to be reduced.

In philosophy this sort of reduction has come to be known as reductive explanation.⁸ Kim (2008) traces its philosophical history as follows.

- He credits (Fodor 1974) with introducing the idea, charactering it as bold and revolutionary.
- He credits Chalmers (1996) with reviving the idea but not taking it far enough.
- He finds the concept still new and fresh enough in 2008 to devote a fair amount of space to fleshing it out as what he calls “functional reduction.”

Reductive explanation = model building = emergence

It should be clear that reductive explanation is another way of describing modeling an abstraction by writing software. In other words, reductive explanation—although not by that name—is standard computer science practice. Virtually every computer program—at least every computer program that was written with some intent in mind—can be conceptualized at a level different from its individual instruction sequences, i.e., at the level of the autonomously specifiable abstractions that it implements. Consequently every computer program is a reductive explanation of those abstractions.

This perspective carries over to what I have been calling the computer science approach to emergence. Just as computer scientists are able to construct new abstractions by combining existing ones in creative ways, so too nature is able to construct new abstractions by combining existing ones in creative ways. That, of course, is a bit of an overstatement. Since nature does not create with intent nature’s creations cannot always be described as autonomous abstractions.

Which one can be, and which ones persist?⁹ Typically it is those that interact successfully with their environments. For something to interact successfully with its environment means first that it is not destroyed by its environment and perhaps second that it can exploit its environment to

⁸ Bechtel (2007) refers to this sort of approach as mechanistic explanation, and Howard (2007) calls it taking a semantic view.

⁹ One of the key issues is ontological: what entities exist, or can they all be reduced away? See Abbott 2009 for further discussion.

persist, e.g., by extracting energy from the environment.

For something to be able to interact successfully with its environment requires that it satisfy some of what might be called the environment's implicitly specified abstractions. There isn't room here to explore the notion of an explicitly specified abstraction in detail, but a good example is the evolution of salt receptors discussed earlier. In evolving salt receptors, biological organisms satisfied the NaCl binding abstraction that existed in the environment.

It is such a constructive approach to emergence that explains the richness and variety that we see around us. Thinking about emergence from a reductionist perspective just seems to miss the boat.

Reduction and concept grounding

The difference between traditional reduction and model development reflects a difference in how (or whether) concepts are grounded. Traditional reduction maps theories to theories. One never grounds either the reducing theory or the theory to be reduced. Model development is grounded on the model side. The model becomes the concrete semantics that grounds the abstract specification for which it serves as a model.

Software allows computer scientists to externalize and ground our concepts as software models. But more importantly—and perhaps paradoxically—the ability to ground our concepts frees us to be that much more creative and to imagine fanciful new abstractions. We can grant ourselves that freedom because we know that any abstraction we imagine will be subjected to the ultimate test: not is it right but does it work?

All creative disciplines ground their concepts: engineering as physical devices, the creative arts as artistic products, etc. As far as I know, though, no other discipline has the means to ground its concepts as symbolic models. Philosophy in particular would probably benefit from a similar grounding mechanism for its concepts.

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 (2007) "Bits don't have error bars," *Workshop on Engineering and Philosophy*, October 2007. To be included in the selected papers from the conference, which is yet to appear.

http://cs.calstatela.edu/wiki/images/1/1c/Bits_don't_have_error_bars.doc

Abbott, Russ (2008a) "If a tree casts a shadow is it telling the time?" *International Journal of Unconventional Computation.*, (4, 3), 195-222. Preprint:

http://cs.calstatela.edu/wiki/images/6/66/If_a_tree_casts_a_shadow_is_it_telling_the_time.pdf

Abbott, Russ (2008b) "Abstraction abstracted," *International Conference on Software Engineering: Proceedings of the 2nd international workshop on The role of abstraction in software engineering*, pp 23-30. Association for Computing Machinery. Preprint:

http://cs.calstatela.edu/wiki/images/5/56/Abstraction_abstracted.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.

Bechtel, W. (2007). "Reducing psychology while maintaining its autonomy via mechanistic explanation." In M. Schouten and H. Looren de Jong (Eds.). *The Matter of the Mind*. Oxford: Basil Blackwell. Preprint: <http://mechanism.ucsd.edu/~bill/research/reducingpsychologywhilemaintainingautonomy.withfigures.pdf>.

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

Byrne, Alex (2006) "Intentionality." *Philosophy of Science: An Encyclopedia*, ed. J. Pfeifer and S. Sarkar, Routledge, 406 – 410. Preprint: <http://mit.edu/abyrne/www/intentionality.html>.

Chalmers, David (1996) *The Conscious Mind*, Oxford University Press.

Fodor, Jerry (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.

Harnad, Steven (1990) "The Symbol Grounding Problem," *Physica D* 42: 335-346.

Hodges, Wilfrid (2008) "Model Theory", *The Stanford Encyclopedia of Philosophy (Fall 2008 Edition)*, Edward N. Zalta (ed.), <http://plato.stanford.edu/archives/fall2008/entries/model-theory/>.

Howard, Don (2007) "Reduction and Emergence in the Physical Sciences." In *Evolution and Emergence: Systems, Organisms, Persons*. Nancey Murphy and William R. Stoeger, S.J., eds. Oxford University Press.

<http://www.nd.edu/%7Edhoward1/Reduction%20and%20Emergence.pdf>

Kim, Jaegwon (2006). "Emergence: Core ideas and issues" *Synthese* 151, 547-559.

Kim, Jaegwon (2008). "Reduction and Reductive Explanation: Is One Possible Without the Other?" in Hohwy, Jakob and Jesper Kallestrup, *Being reduced*, Oxford University Press, 93-114.

Nagel, Ernest (1961) *The structure of Science*, Harcourt, Brace, and World.

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

Searle, John R. (1980) "Minds, Brains, and Programs," *Behavioral and Brain Sciences*, 3(3), pp. 417-457. Preprint:

<http://www.bbsonline.org/documents/a/00/00/04/84/bbs00000484-00/bbs.searle2.html>.