PREPRINT. To appear in a forthcoming edition of Unconventional Computation.

If a tree casts a shadow is it telling the time?

Russ Abbott

*Department of Computer Science, California State University, Los Angele*s, Ca, USA

Russ.Abbott@GMail.com

Abstract. Physical processes are computations only when we use them to externalize thought. Entities provide nature with a way to preserve structure over time. We think in terms of entities because they are so central to how the world is. Computation is the performance of one or more fixed processes within a contingent environment. We reformulate the Church-Turing thesis so that it applies to software rather than to computability. When suitably formulated, agent-based computing in an open, multi-scalar environment represents the current consensus view of how we interact with the world. But we don't know how to formulate multi-scalar environments.

Keywords: agents, agent-based, agent-based computation, Church-Turing thesis, Church's thesis, computing, computation, environment, ideas, interaction, interactive computation, models, multi-scalar environment, thought, thought externalization, thought tools, unconventional computation.

1 Introduction

Merriam-Webster defines [1] *computation* as "the act or action of computing," where *to compute* is defined to mean "to determine." Wolfram's MathWorld defines [2] *computation* as "an operation that begins with some initial conditions and gives an output which follows from a definite set of rules." Neither of these definitions offers much insight into what we mean by *computation*. The former defines it in terms of a more general term. The latter seems to define it as any deterministic process that has a beginning and an end. In this paper we explore the meaning of the term *computation.* We focus on the following questions.

- *What is computation?*
- *How can computation be distinguished from other natural processes?*
- *What is the relationship between ideas and computations?*
- *What is the relationship between a computational process and the environment within which it occurs?*
- *What is the relationship between ideas and how nature is organized?*

We draw the following conclusions.

- Physical processes should be considered computation when they can be treated as externalized thought. In particular, we re-interpret the Church-Turing Thesis to mean that software represents how we understand rigorous thought to be expressed.
- In exploring the relationship between thought and computation, we find that (a) the notion of an entity is central to how nature is organized and (b) our intuitive notion of entities corresponds to this organization.

- Computation involves the playing out of fixed processes against a contingent environment. In particular we agree with Wegner [3] that the agent-based model of computation is the right way to think about interaction with an environment.

1.1 Is Google reading my email?

That's the first question in the Google Gmail help center [4]. This question arises because Gmail places ads next to email messages, and the selection of ads is based on the contents of the messages. Google's answer to this question has varied over time. On March 13, 2006, the posted answer was as follows.

> Google computers scan the text of Gmail messages in order to filter spam and detect viruses, just as all major webmail services do. Google also uses this scanning technology to deliver targeted text ads and other related information. *The process is completely automated and involves no humans.* [Emphasis added.]

In other words, Google's computers are reading your email—but no human beings are. That most people find this reassuring illustrates the intuition that *it's what goes on in the mind of a human being that matters to us.*

One might object that if a computer is reading one's email (and storing its contents in a database), a person might read it later. That's quite true, and the fact that only Google computers (and not Google employees) are reading one's email when selecting ads does not guarantee one's privacy. But if no person *ever* reads one's email, then most people will not feel that their privacy has been violated.

After all, email is read by a number of computers as it passes from sender to receiver. No one has ever worried about that. The moment of violation occurs when some living human being becomes consciously aware of one's personal information.

But, one might argue, the kind of reading that occurs when a computer transmits a message along a communication channel is qualitatively different from the kind of reading that occurs when a Google computer determines which ads to place next to a message. The former treats messages as character strings; no meaning is extracted. The kind or reading that Google computers do extracts (or attempts to extract) meaning so that related ads can be displayed.

This raises the question of what one understands by the term *meaning*. That's clearly a larger topic than can be settled here, but our short answer is that the intuitive sense of meaning has something to do with an idea or thought forming in a mind.[1] At this stage in the development of technology, most people don't believe it makes sense to say that an idea has formed in the mind of a computer—or even that a computer has a mind. We may speak informally and say something like "the computer is doing this because it thinks that." But when we say these sorts of things, we are deliberately speaking metaphori-

[1] This is different from the formal semantics sense in which *meaning* refers to a mapping from an expression to a model.

cally.[2] Until we start to think of computers as having minds that have subjective experience, minds in which ideas can form—then most people will feel comfortable with Google's reply that its computers, but no human beings, are reading one's email.

1.2 Thinking and thought tools

If a tree grows in a forest, but no one counts its rings is it counting years? Is it performing an unconventional computation? If a tree grows in a forest but no one knows it's there, is it instantiating the idea of a tree? These questions have the same sort of answers as does Bishop Berkeley's famous question: if a tree falls in a forest with no one around to hear it, does it make a sound?

Berkeley's question seems more difficult than it is because it confuses two senses of the word *sound*. If a tree falls in a forest, it generates (what are called) *sound waves* whether someone is there to hear them or not. But if no one is there to hear the sound, if no being has a subjective experience of the sound, then no sound will be heard. [3]

The same holds for ideas. Like the subjective experience of a sound, the *idea* of a tree exists only as a subjective experience. If no one has that subjective experience, then a tree without anyone knowing about it will not be instantiating the idea of a tree—because to instantiate an idea requires that the idea exist in someone's mind.

Even if one were to grant that the idea of a tree is exactly the right way to describe that particular aspect of nature, that idea exists only as an idea, and it exists only in the mind of someone who is thinking it. Ideas exist only as subjective experience. In saying this we are taking an explicitly anti-Platonist stance: there is no realm outside the mind in which ideas exist on their own.

This is not intended as mystical or profound—just a statement of the brute fact that an idea is something that occurs only in someone's mind. The ideas in this paper exist only in the mind of the author and the minds of the readers as the author and readers are thinking them. These ideas don't exist on the paper or on the computer screens on which these words appear. They don't exist in the computer memory in which these words are stored. Just as an invasion of privacy occurs at the moment when some being-with-a-mind learns something personal about us, an idea exists only when someone is thinking it.

We go to such lengths to make this point because our position is that computations are ideas that have been externalized in such a way that they can be performed by physical processes. When a tree grows rings, it just grows rings. But when one uses that tree-ring growth as a way to count years, i.e., to help oneself work with ideas—such as the idea of

[2] We are taking what Dennett calls the intentional stance. Although computers don't (and given current technology can't) take an intentional stance, our attributing such a perspective to them reflects what we will refer to later as our externalization of though.

[3] Berkeley answers his question by arguing that the tree makes a sound because God, who is always everywhere, hears it.

a year—then one can say that the tree has performed a computation—an unconventional one.

When a computer runs is it computing? Our answer is the same. A computer is computing only when it is understood to be performing some externalized mental activity. Otherwise, it's just an arena within which electrons are moving about.

1.3 To come

Section 2 continues the discussion of thoughts and introduces the notion of thought tools, for which it provides a brief history. Section 3 steps back from the relationship between ideas and computing and discusses entities as fundamental to nature. Section 4 build on the discussion of entities to discuss thought externalization today and how ideas tend to be the top-down reflection of a bottom-up nature. Section 5 considers how *computation* might be defined. Section 6 discusses the agent-based computing paradigm as more than just an approach to programming and modeling but as common to many of the ways we think about both thinking and our interaction with nature.

2 Historical tools for the externalization of thought

This section sketches a brief history of thought externalization.

2.1 Time computers

Historically we, i.e., humankind, have used natural processes to help us externalize and express our ideas about time, i.e., the daily, monthly, and yearly cycles of the earth, moon, and sun. Not to beat this point into the ground, *day, month,* and *year* are ideas. As ideas, they exist only in the mind—no matter how accurate or true they are as descriptions of nature.

The first time-computers were the natural processes that corresponded to our thoughts. The rising and setting of the sun were the physical events that we used to keep track of the mental events: the *start* and *end* of a *day*. Similarly for the moon. Yearly events such as river floodings and the comings and goings of the seasons helped us keep track of the mental event: the yearly cycle.

This is a somewhat subtle point. Our ideas about time presumably resulted from our observations of the events referred to above. The only reason we thought about *a day* was because of the daily cycle of the sun. But once we invented the idea of *a day*, we turned the tables on the underlying phenomena and used the sun's rising and setting to represent that idea. Reality became for us the embodiment of our ideas.

It didn't take us long to invent more sophisticated means for tracking time. The sundial, for example, is an analog computing device. The position of the sun's shadow is an analog for the mental event *time-of-day* which corresponds to the physical relationships between the relative positions of the sun and the earth.

With the sundial we started to arrange physical materials to help us track our thoughts. In building sundials we set up shadow casters, which in conjunction with the sun and the markings we made on the surface on which the shadow is cast, helped us track (our ideas about) the passing of the day. Presumably building our own shadow casters was a fairly easy step from using pre-existing shadow-casters, e.g., trees, for the same purpose. Hence the title of this paper: if a tree casts a shadow, is it telling the time?

2.2 Number and space computers

Number computers. Apparently we started to count quite early. Bones with notches carved into them appeared in western Europe 20,000 to 30,000 years ago. There is evidence of the use of a tally system—groups of five notches separated from each other. With tally systems not only did we (i.e., humankind) mark physical materials to help us keep track of numbers (which are also mental events), we also invented ways to make counting easier by the way in which we arranged these marks, i.e., in groups. Soon we invented the abacus.

With these primitive computers we freed computation from a dependence on natural processes. Sundials and astronomical masonry depend on the sun and the stars. Counting depends on nothing other than human activity. Once we invented computational devices that were independent of non-human physical processes it was a short step to written notation. By approximately 3,000 BC cuneiform writing on clay tablets using positional notation was known in Babylonia.

Space computers. Besides time and numbers, the Pythagoreans in Greece and Euclid in Egypt developed ways to think about space. Early geometers were concerned about construction issues. The straight edge and compass were their (human-powered) thought tools. They used them to externalize, to create representations of, and to manipulate the ideas of straight lines and circles.

Is it reasonable to call abaci and geometers' tools computers? Even though abaci and geometers' tools depend entirely on human activity to make them "run," it seems reasonable to call them computers because they are used according to mechanical rules. Even though the source of energy for an abacus is the user, the abacus user follows strict rules—rules which could be automated.

2.3 Thought tools for symbol manipulation

Beyond time, numbers, and space, we (human beings) also built thought tools to represent symbolic thoughts and relationships. Sowa [5] describes the Tree of Porphyry as follows.

> The oldest known semantic network was drawn in the 3rd century AD by the Greek philosopher Porphyry in his commentary on Aristotle's categories. Porphyry used it to illustrate Aristotle's method of defining categories by specifying the *genus* or general type and the *differentiae* that distinguish different subtypes of the same supertype.

Another attempt to externalize symbolic thought has been credited to Ramon Lull in the late 13th century. The Smart Computing website [6] describes it as follows.

Ramon Lull’s logic machine consisted of a stack of concentric disks mounted on an axis where they could rotate independently. The disks, made of card stock, wood, or metal, were progressively larger from top to bottom. As many as 16 words or symbols were visible on each disk. By rotating the disks, random statements were generated from the alignment of words. Lull’s most ambitious device held 14 disks.

The idea for the machine came to Lull in a mystical vision that appeared to him after a period of fasting and contemplation. It was not unusual in that day … scientific advances to be attributed to divine inspiration. He thought of his wheels as divine, and his goal was to use them to prove the truth of the Bible. …

In “Gulliver’s Travels,” Swift satirizes the machine without naming Lull. In the story, a professor shows Gulliver a huge contraption that generates random sequences of words. Whenever any three or four adjacent words made sense together, they were written down. The professor told Gulliver the machine would let the most ignorant person effortlessly write books in philosophy, poetry, law, mathematics, and theology.

This may be the first use of non-determinism in computing.

Soon thereafter William of Ockham discovered the foundations of what were to become De Morgan’s laws of logic. More specifically, from Sowa [7]:

(Ockham, 1323) showed how to determine the truth value of compound propositions in terms of the truth or falsity of their components and to determine the validity of rules of inference … in terms of the truth of their antecedents and consequents.

3 Entities are nature’s way of having and remembering ideas

In this section we step back from thought externalization to discuss what the thoughts that are being externalized might be about. In particular, we discuss entities and the relationship between entities and ideas. We define what we mean by *entity* in the following section. Our conclusions will be as follows.

- Entity formation, i.e., the creation of naturally occurring entities, is an objectively real phenomenon by means of which nature creates entities with new properties and functionalities.
- To a great extent, idea formation is a parallel process by means of which we (i.e., human beings) create concepts that correspond to real or imagined entities and their properties.

We as human beings create ideas as a way both to understand nature and to build upon it.

- When we create ideas and attempt to match them to reality we are doing science.
- When we create ideas and attempt to match reality to them we are doing engineering.

3.1 Entities

As discussed elsewhere [8] and [9] there are two kinds of entities: static and dynamic.

- Static entities—for example, atoms, molecules, and solar systems—maintain their structure (and hence their reduced entropy) because they exist in energy wells—and hence have less mass as an aggregate than their components considered separately.[4]
- Dynamic entities—for example, biological organisms, social and political organizations, and (strikingly) hurricanes—maintain their structure (and hence their reduced entropy) by using energy they import from outside themselves—which makes them (famously) far from equilibrium. Because of the flow of imported energy, dynamic entities have more mass as an aggregate than the combined mass of their components considered separately.[5]

3.2 Entities and specifications

Entities have what are often called emergent properties, which are defined at the level of the entity itself. That a government (a dynamic entity) is democratic or that a diamond (a static entity) is hard are properties defined at the level of the government or the diamond. They are not properties of the components of a government or diamond.

Describing something in terms of its externally observable properties is common in both software and systems engineering. In computer science, describing something independently of its implementation is called a specification. The specifications of abstract data types and early attempts to axiomatize software are early examples. It is now commonplace to write specification documents to describe desired software systems prior to building them. Software specifications may be formal (i.e., expressed in a formal language—which is very difficult to carry out in detail) or informal (i.e., expressed in a natural language—which is common practice) as in a natural language specification of a software system's API.[6] Software specifications describe the behavior of software without prejudicing its implementation.

---

[4] Paul Humphreys [10] suggested a similar notion, which he called *fusion*. The following is Timothy O'Connor's summary [11] of Humphreys' position.

> "[Emergent properties] result from an essential interaction [i.e. fusion] between their constituent properties, an interaction that is nomologically necessary for the existence of the emergent property." Fused entities lose certain of their causal powers and cease to exist as separate entities, and the emergents generated by fusion are characterized by novel causal powers. Humphreys emphasizes that fusion is a "real physical operation, not a mathematical or logical operation on predicative representations of properties."

[5] Speaking poetically one might refer to the energy flowing through a dynamic entity as its soul or spirit. When the energy stops flowing, the entity dies. From this perspective a soul or spirit has mass.

[6] An Application Programming Interface (API) is the collection of operations that may be performed on a software system via calls to the system by other software. For each API element one specifies how that element may be called, what parameters it accepts, the effects of the call on both the system (i.e., how the system's conceptual model will be effected by the call) and the parameters, and the results returned if any.

In systems engineering, the description of a system in terms of its observable properties is called a requirements specification—again a description of a system in terms that do not constrain the implementation of those properties.

Familiar as we—as software and system developers—may be with using specifications to describe software or engineered systems, it may nevertheless seem strange to talk about naturally occurring entities such as diamonds or biological organisms in such an abstract way. One wonders how it is possible to discuss the properties of an entity independently of its components. Doesn't its internal organization matter? Do such entities spring into existence fully formed? How can something that seems altogether new—like a bird—and that has new properties—like the ability to fly, a property that seems to be defined in terms of the entity itself—appear apparently from nowhere?

Because this seems so mysterious, one may be tempted to look for hitherto unknown mechanisms for self-organization—or in desperation even intelligent design. This is a distraction. There is nothing mysterious about how entities form. Naturally occurring static entities form as a result of well understood physical laws: atoms are created from elementary particles; molecules form from atoms; etc. Naturally occurring dynamic entities also form as a result of natural processes. Governments form when people create them—either explicitly or implicitly. Hurricanes form when the atmospheric conditions are right. Self-organization is not the point.

The marvel of entities is not in some seemingly magical process of self-organization; the marvel is that entities exist at all and that they have properties and behaviors that may be described independently of their implementations.

3.3 Entities, their properties, and naturally occurring designs

If the question is where do the new properties that we attribute to entities comes from, the answer is that these "new properties" are really nothing more than ideas in our minds. Properties as ideas don't exist in nature. Entities are what they are no matter what properties we attribute to them. The idea of a property doesn't exist in the mind of nature. Nature doesn't have a mind.

This is not say that an entity's new properties are fictitious. Hemoglobin, for example, can bind to, transport, and release oxygen. This property, while true of hemoglobin, is not a label one finds attached to hemoglobin molecules. There is no little FTC-approved[7] tag attached to each hemoglobin molecule that says: certified capable of carrying oxygen. Yes, hemoglobin carries oxygen. But the conceptualization of hemoglobin as having the property of being able to carry oxygen is an idea in our minds, not in some universal mind that tracks the properties of all entities.

[7] The Federal Trade Commission (FTC) regulates commerce in the United States.

Nonetheless, hemoglobin does carry oxygen. And because hemoglobin carries oxygen, a certain form of life—creatures like us—was able to establish and maintain itself on earth.

3.4 Designs and levels of abstraction

Suppose a government hired a contractor to tell it how the country actually worked—from the government on down. The contractor has been asked to produce a complete engineering design description of the country as it currently exists. Presumably, the description would have to include the equivalent of engineering drawings of the government, the components of the government, the components of those components, etc. Since human beings are components of the government, we would eventually find ourselves having to describe the role of hemoglobin molecules in human biochemistry.

Each level of such a description would correspond to what in computer science is called a level of abstraction. For the sake of this example, lets suppose that hemoglobin molecules are black-box components—i.e., biological piece parts—which can be included in our design without our having to build them ourselves. All we care about is their functionality, i.e., their ability to carry oxygen. Thus all one cares about with respect to a hemoglobin molecule is its specification, not how it implements the functionality described by its specification.

Similarly, when describing how the government functions, one would take the description of the kinds of things that people can do as opaque. As far as the government's functioning is concerned one doesn't care that people keep themselves alive through the use of hemoglobin molecules but whether legislators are able to cast votes, for example. Even if in describing a government one were responsible for describing the design of the people who participated in it—and hence had to understand the role of hemoglobin in keeping people alive—when thinking about the functioning of the government itself, one would not be concerned with that aspect of how people are designed.[8]

The important point here is that the design of one level of abstraction, e.g., a government, is expressed in terms of other levels of abstraction, e.g., people, whose designs are expressed in terms of still other levels of abstractions, e.g., hemoglobin. But each level of abstraction can (and should) be documented separately, in terms of other abstractions. This is in contrast to the alternative whereby one would describe the top level design in terms of the lowest level elements. In this case, it would mean that the design of a government would be expressed in terms of biological piece parts such as hemoglobin. Clearly that makes no sense. The structure of a government is defined in terms of roles that are filled by human beings, not by collections of biological piece parts. One can't explain the role of legislators in voting on prospective laws by talking about biological piece parts.

[8] This, of course, is a gross simplification. Governments are concerned, for example, with issues of air quality, which cannot be understood without knowing that human beings rely on oxygen to survive. This is one reason that modeling is so problematic.

Designs of all naturally occurring entities[9] when given in terms of such increasing levels of abstraction are bottom-up designs. The entities at the level below any given level already exist. One is interested only in how they come together to accomplish what they do at the next higher level.

Although the properties and abstractions of naturally occurring designs are not made explicit by nature—nature doesn't document her designs—as they would be were they documented by well-trained engineers, it's clear that nature's designs are best understood in terms of such levels of abstraction.

- A wolf pack is a pack of wolves, not an aggregation of wolf organs and other biological piece parts.
- A wolf is a system of organs and other elements, not an aggregation of molecules and atoms.
- Hemoglobin is a structured pair of proteins and other components, not an aggregation of elementary particles such as electrons and quarks.

Nature is an engineer whose design make sense only when understood in terms of multiple levels of abstraction. Nature accomplishes this feat through the use of entities. It is entities that carry identity, properties, and functionality. But entities are not labeled as such, and entities do not have tags attached to them describing their properties and functions. By embodying what we term properties and functions entities serve as a means for nature to fix and record levels of abstractions—which can be used to build higher levels of abstraction. In other words, entities are how nature remembers its constructions.

3.5 The role of entities in naturally occurring designs

Another way of putting this is that entities are characterized by reduced entropy. Reduced entropy marks a pattern that persists in time. Patterns often have properties and functions.[10] Some properties and functions can be used to build additional patterns. Thus entities are a means by which patterns persist in time, i.e., are "remembered." Patterns that persist in time (and are mutually accessible) are available to be combined to create additional patterns.

The notion of "an entity" is what one might call a design meta-construct. Like a class or object in an object-oriented programming language, the notion of "an entity" refers to a kind of design construct, not to any particular element in any particular design. As a de-

[9] We count governments as naturally occurring entities. Governments may be understood as more sophisticated versions of long-standing naturally occurring animal groupings such as flocks, herds, tribes and (bacterial) colonies.

[10] In section 3.3 we said that properties as ideas don't exist in nature. Patterns certainly have properties (and can perform functions) that we can describe. It is these descriptions (and the names we apply to these descriptions) that exist only in our minds.

sign meta-construct notion of “an entity” plays multiple important roles in naturally occurring designs. As the following paragraphs elaborate entities (a) allow nature to build levels of abstraction; (b) provide nature a way to preserve patterns over time; and (c) serve as nature’s memory.

Entities allow nature to build levels of abstraction. Once a level of abstraction has been constructed (as an entity), nature can then build new levels of abstraction by combining existing levels of abstraction and exploiting their properties and functionality. As indicated above, it simply makes no sense to speak, for example, of a colony of bacteria as if it were a colony of cell organelles and other cellular elements. In order for nature to build the level of abstraction colony-of-bacteria, nature first had to build the bacterium level of abstraction.

So even though nature does not label her levels of abstractions the way we as human designers do—there are no tags saying “bacterium” attached to bacteria—the levels of abstractions and the properties and functionalities that they implement are real nevertheless.

As argued above, a level of abstraction is a specification, a description of something from a behavioral and external perspective. Another way of putting it is that a level of abstraction is a specification (or conceptualization) of a set of properties and functionalities. Informally we might refer to such a conceptualization as an idea. In this sense, entities are nature’s way of having an idea. Put another way, if nature had a mind, entities are what it would use to externalize its ideas.

Entities preserve useful patterns of relationships over time. Organizing two or more entities into a structure of some sort often creates new functionality. Hemoglobin, for example, consists of two strands of protein. They must be combined into a larger organization before they can transport oxygen. An entity is such a persistent stable structure of components.

Entities serve as nature’s memory. If we think of memory as the ability to retain structure, i.e., reduced entropy, over time, entities provide that function for nature. Both static and dynamic entities have reduced entropy (are more structured) than their components would have been otherwise. The creation of an entity is the creation of a means whereby reduced entropy persists over a period of time.

Consider the difference between the face one may see in a cloud and a similar face on a human being. The face in a cloud is fleeting; no mechanism exists to retain it. The face of a living human being persists. It changes as the person changes, but it persists as a face over time. Entities with their built-in mechanisms for persistence provide a way for nature to retain structures that are imposed over the elements that make up the entity.

Static entities impose structures over fixed collections of components. Dynamic entities impose structures over changing collections of components. The atoms in the body of a biological organism change over time. The members of a social organization change over time. The citizens of a country change over time. With the development of dynamic entities nature created a way to remember structures which are separate from the components that the structures organize—quite a trick.

3.6 The reductionist blind spot

Isn't it obvious that higher level entities are composed of lower level entities? Why even bother to say that a flock of birds consists of birds? Extreme reductionism claims that all explanations can be formulated in terms of the fundamental laws of physics as they pertain to elementary particles. Steven Weinberg [12] puts it this way.

> [Reductionism is] the view that all of nature is the way it is (with certain qualifications about initial conditions and historical accidents) because of simple universal laws, to which all other scientific laws may in some sense be reduced. …
>
> Every field of science operates by formulating and testing generalizations that are sometimes dignified by being called principles or laws. … But **there are no principles of chemistry that simply stand on their own, without needing to be explained reductively from the properties of electrons and atomic nuclei** [emphasis added], and in the same way there are no principles of psychology that are free-standing, in the sense that they do not need ultimately to be understood through the study of the human brain, which in turn must ultimately be understood on the basis of physics and chemistry.

It is this view that the notion of entities developed in this paper disputes. Consider two examples: a solar system and a living biological organism. We claim that neither can be understood strictly in terms of the principles of physics.

For one things, neither can even be defined in terms of the principles of physics. How would one define *solar system* in a definition (or a cascade of definitions) that contained references to nothing but elementary particles and forces? A solar system is not just a collection of elementary particles under mutual gravitational attraction. A solar system consists of one or more *stars* along with one or more *planets* orbiting around that (or those) *stars*. But what is a *star*, and what is a *planet*? Neither can be defined without implicitly or explicitly building in the notion of an entity, i.e., patterns that persist in time.

Furthermore, if one talks about proprieties of a solar system, such as the number of its planets, or the length of the year of one of its planets, or whether the orbit of a planet is chaotic, etc., those ideas also rely on the notion of a planet as an entity.

Certainly, stars and planets are made up of elementary particles, and certainly it is the force of gravity, an elementary force, that holds it all together. But it is wrong to say that notions such as a solar system are reducible to terms defined at the level of elementary physics.

This is not playing with words. The very notion of a solar system is built on the notion of a star and some bodies orbiting it. If one can't talk about those bodies as entities, the notion of a solar system has no meaning.

The case for biological organisms is even more striking. How would one define the term *alive* using concepts from elementary physics? In our view it makes sense to define *alive* as a property of dynamic entities. A dynamic entity is alive as long as it persists. But of course, unless one includes the notion of entities, and especially dynamic entities, within the realm of elementary physics, that sort of definition is not accessible to the pure reductionist.

At a more concrete level, how would one discuss the mechanism through which oxygen-breathing organisms keep themselves alive? To do so, one must talk about hemoglobin and oxygen molecules, i.e., about entities. The requirement that oxygen be carried from the lungs (how are lungs defined in terms of elementary physics?) to the rest of the body and the story of how that is accomplished can't be told in terms of elementary physical particles. It's not a matter of quarks, electrons, etc.

3.7 Entities are real, but forces and causes are epiphenomenal

All the entities involved in describing the role of hemoglobin in keeping biological organisms alive are made up of elementary physical particles. And all the forces involved are elementary physical forces. But the description of the design of biological organisms as dependent on the property of hemoglobin to transport oxygen simply is not a description that can be told in the language of elementary physics. Isn't this a contradiction?

We are not claiming that the particles and forces of elementary physics are not relevant to either solar systems or biological organisms. They are essential. As we discussed in [9] forces and causality (such as the motion of a hemoglobin molecules along the blood stream) that one might like to attribute to entities (such as the heart) found on levels higher than that of elementary physics are epiphenomenal—i.e., they appear to be separate from their underlying causes but are really just a reflection of those causes seen in a different way. There are no higher level forces: there is no vital force. But if there are no higher level causes—if all apparently higher level causes are epiphenomenal—how can we make the claim that higher level entities are both real themselves and real elements of even higher level designs?

The answer is that higher level entities are real but that interaction among them is epiphenomenal. There are only primitive physical forces. But because entities exist, the patterns that they represent organize those forces to create functionalities that are best described at the level of the entities. Consider hemoglobin again. Hemoglobin transports oxygen. One can't just say that an aggregation of elementary particles that make up oxygen-bound-to-hemoglobin is carried along. Furthermore, one must talk about the heart as the source of power that pumps the hemoglobin carried in a stream of fluid through the body along a network of arteries and veins. None of that can be described in the language of elementary physics without implicitly or explicitly importing the notion of entity and their derived functionalities.

Contrary to Weinberg we claim that it is reasonable to describe the functioning of oxygen-breathing organisms as principles of biology because it is at the level of biological entities that biological functionalities come into being. The mechanisms used at the biological level are separate from and cannot be reduced to those of elementary physics. These are mechanisms that are described on the level of blood vessels, oxygenation, pumps, lungs, hemoglobin, etc. The structures that have been built to support oxygen-breathing organisms are new creations in much the same way that the algorithms built into computer software are new creations.

Certainly the principles of biology must be implemented by mechanisms that operate on the level of elementary physics. But one could never derive the fact that biological organisms depend on hemoglobin-carrying oxygen being pumped though the body from the principles of elementary physics. Implementation of the laws of the higher level sciences by those of elementary physics is not the same as reduction of those laws to the laws of elementary physics. (We discuss the difference between *implemented by* and *reducible to* in the following section.)

It is entities that serve as the ontological components in terms of which the laws of the higher level sciences are expressed. Entities are physically real, and entities obey laws that must be implemented by—but cannot be reduced to—those of elementary physics. The reductionist blind spot is the failure to see and understand the reality and significance of entities. The reductionist blind spot derives from the confusion caused by the fact that although entities are objectively real, interactions—i.e., forces and causal relationships—among higher level entities are epiphenomenal.

3.8 Patterns are implemented by but are not reducible to the elements they organize

The notion of *implemented by but not reducible to* deserves some attention. A computer program is implemented by the operations defined by the programming language in which it is written. But the functionality of the computer program is not reducible to those operations. Although an algorithm is composed from a set of basic operations, it is neither derivable from nor a logical consequence of those operations. Similarly a musical composition is implemented by the notes of the scale. But the melodies and harmonies of a musical composition are neither derivable from nor reducible to the notes themselves.[11]

In these cases, as in nature, raw materials and fundamental operations are organized into specific patterns to create something new. The patterns built into such designs are separate from and not reducible to the components and forces that those patterns arrange.

As noted above one of the roles that entities play is that they are nature's way of preserving patterns over time. As the examples have illustrated, elements arranged in a pattern often have properties that are separate from the properties of the underlying elements. These pattern-level properties are often not even describable in the language used to de-

[11] Although algorithms and musical compositions are important (or useful or enjoyable), neither are entities according to our definition. Algorithms and musical compositions don't persist on their own. Even though all entities embody patterns not all patterns are entities.

scribe the underlying elements. (One can't describe what it means for a person to breath if one restricts oneself to terms from elementary particle physics.) This may seem profoundly obvious, but it seems to be a point that people tend to forget.

4 Thought externalization, science, engineering, and computer science

4.1 Science is the reverse engineering of nature

Science is a search for an explanation of how nature works. We observe phenomena, which we attempt to describe in terms (ideas) that fit the phenomena. Much of early biology and chemistry followed this pattern. These disciplines organized and catalogued biological and chemical entities into the well known biological taxonomies and periodic table of chemical elements respectively. In effect we developed specifications of the phenomena that we observed. Once we have such a specification, we (as scientists) look for underlying mechanisms that we hope will explain how nature brings about the specified phenomena. In other words, science is the reverse engineering of nature.

4.2 Defining entity*: converting a phenomenological definition to a physical definition*

There is no generally accepted definition of entity in the philosophical literature. In [9] (and above) we define *entity* in physical terms as a persistent phenomenon, i.e., a pattern with either more or less mass and with lower entropy than the elements in the pattern would have on their own. In offering this definition, we are following what has become standard practice in science: converting a phenomenological or intuitive definition into a more physical one. Here are two examples.

In his review of the development of the periodic table Scerri [13] points out that chemists originally thought that chemical elements were characterized by their atomic weights: gold is different from iron because it is more dense. We now know that it is the number of protons that characterizes a chemical element. Thus the intuitive, phenomenological, and informal idea of a chemical element—as a particular type of matter that has certain chemical properties—was made precise by understanding that atomic substances are best grouped according to the number of protons they contain rather than (what we now call) their atomic weight.[12]

[12] Atomic weights: Iron: 55.8, Gold: 197. Atomic numbers: Iron: 26; Gold: 79. This information is based on Google searches as of June 4, 2007.

What is remarkable about these searches is that the desired information was returned directly on the search page. Google did better than find web pages with the search words. It returned the information itself.

It will do just as well with the search: President of France. In this case, however, it was wrong. It reported that Jacques Chirac is the president of France, citing https://www.cia.gov/library/publications/world-leaders-1/world-leaders-f/france.html. In fact, at this writing Nicolas Sarkozy is the President of France. Google relied on a source that was not up to date.

In a completely different realm, we, i.e., human beings, now think of a year as the time it takes the earth to make a complete orbit around the sun—not a cycle through a sequence of weather periods or a certain number of days.

In both of these examples, ideas that started out as specifications of observable phenomena (a chemical element is a class of identically behaving substances, and a year is a traversal though a one cycle of a weather pattern) were redefined in terms of lower level phenomena that could be seen as implementing the observed phenomena. A chemical element is now *defined* in terms of protons, and a year is now *defined* in terms of the orbit of the earth around the sun.

These examples all illustrate the externalization of thought. An idea about which we had a basic (phenomenological or intuitive) sense (a chemical element, a year, an entity) is externalized and attached to something concrete in the world. Our defining *entity* as a physical patterning of matter follows along the same path.

4.3 Engineering as thought externalization

Engineering, especially the engineering of large systems, may be understood as something like the externalization of an imagined construct. As human beings we think, "I want a system that does this, this, and that—i.e., with these properties and behaviors." Like all thoughts, dreams of this sort, no matter how dressed up and legitimized in terms of formal requirements are nothing but ideas in our minds.

Yet when our ideas involve imagined systems, we want more than just pretty mental pictures. We want material embodiments of our ideas. We want to have the ideas in our minds converted into physical reality. We want to externalize our ideas and to make them materially concrete. And we often succeed—spectacularly. Much of what we experience in our post-modern 21st century lives is the result of successfully externalized dreams.

But let's consider what it means to externalize a thought of something that doesn't exist. There is no *externalize* button on our foreheads that, when pressed, causes ideas to materialize as physical reality. Furthermore, even when we as engineers build something that reflects our ideas, it is impossible to create an external replica of a thought. Nothing outside our heads is a thought. The best we can ever do in externalizing a thought is to create something that we can understand as representing—or perhaps embodying—that thought.

Consider a word processing computer program. The software industry has designed word processors to (appear to) operate in terms of characters, words, paragraphs, etc. Characters, words, and paragraphs are ideas. Word processors operate (when described at one reasonable level of abstraction) in terms of character codes, sequences of character codes bounded by white space character codes, and sequences of character codes bound together as what the word processor may internally refer to as a paragraph data structure.

What we as system developers do when we attempt to externalize an idea is to mold elements of physical reality into a form that we can imagine as embodying the idea we want to externalize. That's all we can ever do. We can never do more than mold existing reality. But even though we cannot incarnate our ideas as material reality, we can mold physical reality in such a way that it has—or at least appears to have—properties a lot like those of the ideas we want to externalize.[13] Thus there is always a gap between (a) building something out of real physical materials (even if those materials involve bits) and (b) externalizing one's thoughts about what one wants.

This gap is easiest to describe with respect to software—but it is true of every constructive discipline, including systems engineering. When one writes software, one is writing instructions for how a computer should perform. That's all one can ever do: tell a computer first to do this and then to do that. The *this* and *that* which the software tells the computer to do are the computer's primitive instructions. But what we want in the end is for the computer's *this-ing* and *that-ing* to produce a result that resembles some idea in our minds—or our user's minds.

Thus in software (as in any engineering discipline) creations always have two faces: (a) a reality-molding face whereby the software (or the engineering design) tells the computer (or other material substance) what to do and (b) a thought externalizing face which represents the ideas that characterize what we want the result of that molding process to mean. The trick is to make these two faces come together in one artifact.

In much the same way as science tends to find bottom-up definitions for what start out as top-down ideas, the two faces of software and engineering creations often come together when an implementation comes to define the conceptual. Most likely we will all soon think of *paragraph* as meaning whatever MS Word manipulates—although many of us will continue to distinguish between paragraphs that are well- and ill-structured.

4.4 Thought externalization in computer science

Every computer application represents the externalization of thought. The thoughts that have been externalized and that are being manipulated are the conceptual model implemented by the application.

Programming languages are tools for externalizing thoughts. Programming languages allows us as software developers to externalize in the form of computer programs our thoughts about symbolic structures and behaviors.

But programming languages are also computer applications. As a computer application, a programming language implements a conceptual model; it allows its users to express their thoughts in certain limited ways, namely in terms of the constructs defined by the programming language.

[13] My wife, an English professor, objected to my claim that word processors don't work with paragraphs. They do such a good job of manipulating text in a way that corresponds to her sense of what a paragraph is, that she wants to credit them with working with actual paragraphs.

All modern programming languages are conceptually extensible. Using a programming language one can define a collection of concepts and then use those concepts to build other concepts. In particular object-oriented programming languages allow their users to create symbolically what nature does when it creates new entities.

As a society we are still learning to use the power of computers to externalize thought. In one way or another, much of software-related research is about developing more powerful, more specialized, faster, easier to use, or more abstract thought tools. The more we learn about externalizing our thoughts the higher we ascend the mountain of abstraction and the broader the vistas we see. Work in thought externalization includes declarative programming (e.g., logic programming, functional programming, constraint-based programming, rules-based systems such as expert systems, etc.), meta and markup languages such as XML and its extensions and derivatives, the Unified (and Systems) Modeling Language (UML and SysML), and the Semantic Web and the OWL Web Ontology Language for externalizing how we look at the world. The developers of OWL are working in a tradition that dates back at least to Porphyry. Individual software applications also represent externalization of how we think about the subject matter those applications cover. Thought tools for the manipulation of images, sounds, videos, etc. have externalized ways of thinking about those domains. Because software can be about an extraordinarily wide range of possible thoughts, computer science has had to confront the reality-vs.-thought gap more directly than any other human endeavor.[14]

4.5 Software as intermediary between thought and reality

Software (as text) seems to be the only example of an artifact that bridges the gap between thought and reality. Software is both a direct expression of thought as well as a means for molding reality. When one writes a computer program one is both expressing one's thoughts and controlling the operation of a computer.

- As computer scientists we invented so-called higher level programming languages (Fortran being one of the earliest) in which one could write something like mathematical expressions which the computer would evaluate.
- We invented declarative languages (Prolog is a good example) in which one could write statements in something like predicate calculus and have the computer find values that make those statements true.
- We combined Prolog and Fortran when we invented constraint programming (which has not been as widely appreciated as it deserves) in which one can write mathematical statements of constraints which the computer ensures are met.

[14] Much of the perspective on entities outlined in Section 3 is simply the application of software development concepts to nature.

- We invented relational databases in which one can store information about entity-like elements—along with their attributes and their relationships to each other.
- We invented languages that allow one to query those databases more or less on the level of that conceptualization.
- We invented object-oriented programming languages—which led naturally to agent-based and now service-oriented environments—in which one writes programs that consist of interacting entities.
- At the application level, virtually *every* computer program—from a payroll program to a word processor to an image processing program—embodies an ontology of the world to which that application applies.
- To help us write application programs we invented tools and frameworks that define meta-ontologies within which one can create a desired ontology.

We did all this by writing programs that tell computers first to execute this instruction and then to execute that instruction. The gap between the underlying computer and the languages in which we write programs is often enormous. But that doesn't mean that one can forget about the computer. No matter what else it is, and no matter how well our programs (seem to) express the thoughts in our minds, a program is nothing unless it tells a computer which instructions to execute and in what order. In the end, that's all a computer program is: a means to tell a computer what to do. A computer program is always a way of shaping reality. But a computer program is written in such a way that the result comes close to embodying ideas in our minds.

Computer programs are prototypical examples of how top-down conceptualizing can match bottom-up reality shaping. One may think of Computer Science as applied philosophy:[15] one can think about virtually anything as long as one can express those thoughts in a form that can be used to control the operation of a computer.

Alternatively, one may think of the computer as a reification machine: it turns symbolically expressed abstract thought into concrete action in the physical world.[16] As a reification machine, the computer's interface between thought and action is the computer program. When we as software developers write in a programming language we are expressing our thoughts in the programming language—to the extent allowed by the language. When a computer reads what we have written, it takes our writings as instructions about what operations to perform. One's hope is that its actions will correspond to the original thought.

[15] Fred Thompson, one of my early mentors, is now Emeritus Professor of Applied Philosophy and Computer Science at Cal Tech.

[16] With virtual reality we complete the cycle: generating real physical signals with the intention of producing particular subjective experiences.

4.6 Thought externalization in systems engineering

Although systems engineering, like computer science, defines itself as the externalization of thought, systems engineering is just beginning to focus on the issue of concrete thought externalization. Model-based development, e.g., SysML, attempts to allow systems engineers to think in a language that both expresses their thoughts and molds at least a virtual reality. But systems engineering is at a significant disadvantage. In computer science developers write in languages that control real computers.[17] There are no systems engineering languages that generate real physical systems. The reality that SysML molds is a virtual reality at best.

When software developers (a) write a computer program, (b) load it into a computer, and (c) press the Start button, the computer *becomes* the program they have written. There is nothing comparable for systems engineers. There is no systems engineering a device into which descriptions written in a systems engineering language can be loaded such that the device will *become* the system the language is describing. The closest systems engineering can come to this dream is to write in a language that represents a model of a physical system. But models aren't reality.

Programming languages succeed because they are grounded in the reality of an actual computer executing actual instructions. Models, in contrast, are always divorced from reality. One can't ever model all aspects of a system. So one chooses what one considers a system's most important aspects and models those. But that's always dangerous. See the discussion in [9] about the difficulty of looking downwards.

5 Defining *computation*

In this section we return to the question of how to define *computation*.

5.1 Defining the term computation

It is surprisingly difficult to find a well considered definition. The one offered by Eliasmith [14] appears to be the most carefully thought out. Here is his definition and his commentary.

> *Computation***.** A series of rule governed state transitions whose rules can be altered.
>
> There are numerous competing definitions of *computation*. Along with the initial definition provided here, the following three definitions are often encountered:
>
> 1. Rule governed state transitions
> 2. Discrete rule governed state transitions
> 3. Rule governed state transitions between interpretable states
>
> The difficulties with these definitions can be summarized as follows:

[17] UML is an unfortunate step away from computer science's traditional loyalty to executable languages.

a) The first admits all physical systems into the class of computational systems, making the definition somewhat vacuous.
b) The second excludes all forms of analog computation, perhaps including the sorts of processing taking place in the brain.
c) The third necessitates accepting all computational systems as representational systems. In other words, there is no computation without representation on this definition.

Contrary to Eliasmith we suggest the following.

a) The notion of alterable rules is not well defined, and hence all physical systems *are* potentially computational systems.
b) But, it is exactly the fact of interpretability that makes a physical process into a computation. (Eliasmith doesn't explain why he rejects the notion that computation requires interpretation.)

Eliasmith requires that the rules governing some identified state transitions must be alterable in order to distinguish a computation from a naturally occurring process—which presumably follows rules that can't be altered. But all computing that takes place in the physical world is based on physical processes. If we set aside the probabilistic nature of quantum physics, and if we suppose that physical processes operate according to unalterable rules, it's not clear what it means to say that it must be possible to alter a set of rules.

This is not intellectual nit-picking. Certainly we all know what it means to say that one program is different from another—that "the rules" that govern a computation may be altered. But the question we wish to raise is how can one distinguish the altering of a program from the altering of any other contingent element in an environment?[18]

It is the particular program that is loaded into a computer's memory that distinguishes the situation in which one program is being executed from that in which some other program is executing. But a computer's memory is the environment within which the computer's cpu (or some virtual machine) finds itself, and a loaded program defines the state of that environment. The cpu (or the virtual machine) is (let's presume) fixed in the same way that the laws of nature are fixed. But depending on the environment within which it finds itself—i.e., the program it finds in its environment—the cpu operates differently, i.e., it performs a different computation. This same sort of analysis may be applied to virtually any natural process. When one puts objects on a balance scale, the scale's behavior will depend on the objects loaded, i.e., on the environmental contingencies.[19] In both the case of programs loaded into a computer and objects put in the pans of a balance scale, we (the

[18] Neither we nor Eliasmith address the issue of "hard-wired" computations. How fixed must state transitions be before one is no longer willing to say they aren't alterable—and hence not a computation?

[19] When a balance scale compares two objects and returns an "output" (selected from *left-is-heavier*, *equal-weights*, and *right-is-heavier*), is it performing a computation? It is if we are using it for this purpose. It isn't if we are using it as a designer setting for flower pots.

user) determine the environment within which some fixed process (i.e., the rules) proceeds.

This brings us back to our original perspective. A process in nature may be considered a computation only when we use it as a way to work with externalized thought. A physical or otherwise established process—be it the operation of a balance scale, a cpu, the Game of Life, or the sun's movement relative to trees and the ground—is just what it is, a fixed process.[20] But for almost all processes,[21] whether we create them or they arise naturally, how the process proceeds depends on environmental contingencies.

When we as thinkers control (or interpret) the contingencies in the environment of some mechanism so that we can use the resulting process to work with our own thoughts, then the process may be considered a computation. That is the case whether we control the contingencies by loading a program into a computer, by placing objects on a balance scale, by establishing initial conditions for the Game of Life, or by using shadows cast by trees to tell time.

Consequently Eliasmith is right in holding that it must be possible to alter a process for it to be considered a computation. That position is better understood as saying that for a process to be considered a computation there must be something contingent about the environment within which it operates that determines both how it proceeds and how we interpret the result.

In other words, we can always separate a computational process into its fixed part and its contingent or alterable part.

- The fixed part may be some concrete instances of the playing out of the laws of nature—in which case the contingent environment is the context within which that playing out occurs.
- Or the fixed part may be the operation of a cpu—in which case the contingent environment is the memory which contains the program that is being executed.
- Or the fixed part may be the operation of a program that a cpu is executing—in which case the contingent environment is the input to that program.

In general a computation occurs when we as thinkers alter the contingencies in the environment of a fixed process as a way to work with our thoughts.

[20] Of course many processes—such as the operation of a cpu and the operation of a balance scale—are what they are because we built them to be that way—because we anticipated using contingencies that we could control in their environment to help us think.

[21] Some quantum processes may occur on their own without regard to their environment—although even they are environmentally constrained by the Pauli exclusion principle..

5.2 Non-algorithmic and unconventional computing

A corollary is that all computation performed by real-world processes are environmentally driven. Computing involves configuring environmental contingencies, i.e., setting up an environment within which a process (or multiple processes) will play themselves out. We refer to this as *non-algorithmic computing* because one's focus is on how an environment will shape a process rather than on a specific sequence of steps that the shaped process will take. No explicit algorithm is involved. Most of what one thinks of as unconventional computation is non-algorithmic in this sense.

It may seem ironic that what one thinks of as conventional computation is a constrained form of unconventional computation. Conventional computation is attractive because its single threaded linearity makes it easy to manage. But nature is not a von Neumann computer. Any computer engineer will confirm how much work it takes to shape nature into a von Neumann computer. Even more ironically, people studying unconventional computation turn around and use conventional single-threaded computers to simulate unconventional computation. One might say that a goal of this journal is to eliminate the von Neumann middle man—to find ways to compute, i.e., to externalize thoughts, by mapping them more directly onto the forces of nature operating in constrained environments.

5.3 Turing Machines and computation

All our definitions of computation so far have relied on human thought. Can one look to Turing Machines (and their equivalents) for a definition of *computation* that is defined independently of thought? But Turing Machines, recursive functions, and formally equivalent models rely on the notions of symbols and symbol manipulation, which are fundamentally mental constructs.

Although Turing Machines and their Church-Turing Thesis equivalents do not help with a thought-independent definition of *computation*, they do offer an important insight. The Church-Turing Thesis identifies symbol manipulation to be what we intuitively think of as computational activity. Thus the Turing Machine model is a way of externalizing an entire class of mental activities, the class that we intuitively identify as computational.

In saying this we are separating (a) the sorts of *computational activities* characterized by Turing Machines from (b) the *class of functions* that these models compute. Computability theory starts with a class of software—the software that one can write by programming a Turing Machines. It then it asks about that class of software which functions can it compute. But the question of which functions one can compute with a Turing Machine isn't the important question. What's important about the Church-Turing Thesis is the possible programs one may write. With the Turing Machine and its formal equivalents the Church-Turing Thesis has characterized a fundamental mode of thought—those thoughts that can be externalized as computations.

Thus our revised version of the Church-Turing Thesis is that to be considered a computation a thought process must, at least in principle, be expressible, i.e., externalizable as software, i.e., as a Turing Machine or equivalent. In making this claim it's important to note that the various equivalent models of computation are all defined constructively. In other words any Turing Machine may be transformed into a recursive function or *vice versa*. Turing Machines, recursive functions, etc. are equivalent as programming languages and not just because they can compute the same set of functions.

6 Agent-based computing

The Turing Machine model is single threaded—as are single processor von Neumann computers. But many thought models are either parallel, asynchronous, or non-deterministic. We as hardware and software designers have not yet worked out a way to deal with complete asynchronicity. But a compromise arose approximately four decades ago in the form of agent-based computing. (See Dahl [15].) Agent-based computing provides an attractive form of asynchronicity because it relies on manageable parallelism—asynchronous computing threads that don't result in an unrealizable demand for computing resources.

6.1 Open and far-from-equilibrium computing

Goldin and Wegner [16] have defined what they called *persistent Turing Machines* (and elsewhere *interaction machines*). These are Turing Machines that perform their computations over an indefinite period—continually accepting input and producing output without ever completing what might be understood as a traditional computation—and not ever necessarily computing a function. Results of computations performed after accepting one input may be retained (on the machine's "working tape") and are available when processing future inputs. Although Wegner's focus is not on agent-based computing, his model is essentially that: agents which interact with their environments and maintain information between interactions. From here on we use *agent* to refer to an object that embodies a program.

Goldin and Wegner claim that their "interactive finite computing agents are more expressive than Turing machines." There has been much debate about this claim. We believe that to ask about the level of computability of agents is to ask the wrong question. We believe that Wegner and Goldin have implicitly taken the same stance that we took explicitly, i.e., to distinguish between the programs one can write and the functions those programs can compute.

Wegner and Goldin point out that one need not think of the program that a Turing Machine embodies in functional terms, i.e., as closed with respect to information flow. One can also think of a Turing Machine as open with respect to information flow. This parallels the distinction in physics between systems that are closed and open with respect to energy flows. Complex systems are famously far from equilibrium with respect to environmental energy flows. Wegner and Goldin's interaction machines (and agents in general) are similarly far from equilibrium with respect to information flows.

What might one gain from being open to information flows? An illustrative example is Prisoner's Dilemma (PD). If one were to develop an optimized PD player for a one-shot PD exchange—since it's one shot, the system is closed—it will Defect. Playing against itself, it will gain 1 point on each side—using the usual scoring rules. If one were to develop an optimized PD player to engage in an iterative PD sequence—the system is open—it will Cooperate indefinitely (presumably by playing a variant of Tit-for-Tat), gaining 3 points on each side at each time. Thus the same problem (PD) yields a different solution depending on whether one's system is presumed to be open or closed with respect to information flows.

6.2 Agents and their environments

Computation involves the interaction of a process with its environment. In all cases with which we are familiar, the environment is modeled as a simply structured collection of symbols, e.g., a tape, a grid, etc. None of these models are adequate when compared to the real-world environment. We do not know how to model the multi-scalar face that nature presents to us—but almost certainly it won't be as a tape or a grid.

- In our actual environment new entities and new kinds of entities may come into existence. We are able to perceive and interact with them. We are aware of no formal environmental framework capable of representing such phenomena.
- We do not understand the ultimate set of primitives—if indeed there are any—upon which everything is built.

We have referred [8] to these problems as the difficulty of looking upwards and the difficulty of looking downwards respectively.

Given our lack of understanding about these issues it is not surprising that we have not been able to develop a formal model of such an environment. Thus a fundamental open problem in computing is to develop a formal model of an environment that has the same sorts of multi-scalar properties as our real-life environment.

Our revised version of the Church-Turing Thesis gives us confidence that our current understanding of agents as entities that embody programs is reasonably close to how we think about thinking. We are still quite far from the goal of formalizing appropriate environments within which such agents should be situated.

6.3 The inevitable evolution and acceleration of intelligence

As we saw in the PD example, thinking in terms of an open computation model leads to different results from thinking in terms of closed models. Yet both use the same class of possible programs—whatever is programmable in a general purpose programming language. Since open computation models include the class of Oracle machines, computability doesn't seem like the appropriate perspective when analyzing these systems. Is there another approach? We suggest that the results achieved is more relevant. In the PD case, the result achieved is the number of points scored.

Unfortunately most agent-based computer models either ignore the issue of energy or treat it very superficially. We believe that an integrated theory of energy and information would clarify how information flows enable evolution. A real-world agent would be a dynamic entity that embodied some software. If, through a random mutation, such an entity developed an enhanced ability to extract information from its environment then it will be more likely to survive and reproduce. What evolves in this model is an enhanced ability to extract information from the environment. The need of dynamic entities for energy drives evolution toward increasingly more powerful informational processing capabilities.[22]

In this picture, information is being extracted from the environment at two levels. Each individual extracts information from the environment, which it processes as a way to help it find energy. Very simple real-life examples are plant tropisms and bacterial tendencies to follow nutrient gradients. More interestingly, the evolutionary process itself extracts information from the environment, which it then encodes (in DNA) as the "program" which individual agents use to process information from their environment. Thus the real intelligence is in the program, and the real information extracting activity is the evolutionary process that constructs the program.[23]

Can evolution itself evolve? Is there something that will enable an entity to extract information from the environment more effectively? Modern society stores information about how to process information from the environment as science.

7 Conclusion

An environmentally sophisticated agent-based paradigm involves agents, each of which has the computing capability of a Turing machine, situated in an environment that reveals itself reluctantly. Such an agent in a real-world environment is like an Oracle machine, with nature as the oracle. Combining agents with dynamic entities yields real-world agents, which (a) must extract energy from their environment to persist and (b) embody software capable of processing information flows from the environment. The agent-based thesis is that this paradigm represents how, at the start of the 21st century, we think about our place with the world.

Acknowledgment. Many of the ideas in this paper were elaborated in discussions with Debora Shuger. I also wish to thank an anonymous reviewer for a careful reading.

References

[1] Merriam-Webster (accessed June 2, 2007) *Online Dictionary.* http://www.m-w.com/dictionary/computation.

[22] This seems to answer the question of whether evolution will always produce intelligence. It will whenever increased intelligence yields enhanced access to energy.

[23] Systems that have attempted to model this process have failed because their environments are too poor.

[2] Wolfram (accessed June 2, 2007) *MathWorld* http://mathworld.wolfram.com/Computation.html.

[3] Wegner, P., Arbab, F., Goldin, D., McBurney, P., Luck, M. and Robertson, D. (2005) "The Role of Agent Interaction in Models of Computing," *Electronic Notes in Theoretical Computer Science*, 141.

[4] Google, *Help Center* (accessed March 13, 2006) http://mail.google.com/support/bin/answer.py?answer=29433&query=faq&topic=0&type=f.

[5] Sowa, J. (accessed June 2, 2006) "Semantic Networks," http://www.jfsowa.com/pubs/semnet.htm, revised from S.C.. Shapiro (ed) *Artificial Intelligence-Encyclopedias*, John Willy & Sons, Inc, 1992, pp 1493-1511.

[6] Smart Computing, (accessed February 20, 2006) http://www.smartcomputing.com/editorial/dictionary/detail.asp?guid=&searchtype=1&DicID=18707&RefType=Encyclopedia.

[7] Sowa, J. (2002, accessed February 20, 2006) "Existential Graphs,". http://www.jfsowa.com/peirce/ms514.htm.

[8] Abbott, R. (2006) "Emergence Explained: Abstractions," *Complexity*, September 2006.

[9] Abbott, R. (2007) "Emergence Explained: Entities," in preparation.

[10] Humphreys, P. (1997) "How Properties Emerge," *Philosophy of Science*, 64, pp. 1-17.

[11] O'Connor, T. and H. Y. Wong (2006) "Emergent Properties", *The Stanford Encyclopedia of Philosophy (Winter 2006 Edition)*, Edward N. Zalta (ed.), http://plato.stanford.edu/archives/win2006/entries/properties-emergent/.

[12] Weinberg, S., (1995) "Reductionism Redux," *The New York Review of Books*, October 5, 1995. Reprinted in Weinberg, S., *Facing Up*, Harvard University Press, 2001. http://pespmc1.vub.ac.be/AFOS/Debate.html. (accessed May 2005)

[13] Scerri, Eric (2006), The Periodic Table: Its Story and Its Significance, Oxford University Press.

[14] Eliasmith, C. (2004) "Computation," entry in *Dictionary of the Philosophy of Mind*, C. Eliasmith (ed.). http://philosophy.uwaterloo.ca/MindDict/computation.html (accessed June 3, 2007)

[15] Dahl, O-J and K. (1966) Nygaard, "Simula: an ALGOL-based simulation language," *Communications of the ACM*, (9, 9), 671-678.

[16] Goldin, D. and P. Wegner (1999) "Behavior and Expressiveness of Persistent Turing Machines," Computer Science Technical Report, Brown University.

http://www.cs.montana.edu/~elser/turing_papers/Behavior_and_Expressiveness_of_PTM.pdf.

[17] Wegner, P. and D. Goldin (1999) "Interaction, Computability, and Church's Thesis," *British Computing Journal*, 1999.