6 Abstraction/Representation Theory and the Natural Science of Computation

Dominic Horsman, Viv Kendon, and Susan Stepney

6.1. Introduction

Is computation an intrinsic property of physical systems, or is there a distinction between a computer and other objects in the universe? Computer science as a theoretical discipline, traditionally dealing only with issues of abstract computation, has tended to ignore this question. Amongst more philosophical approaches to computing, the first view, pancomputationalism (Piccinini 2017), has been argued for in various guises. Assertions such as "the universe is a computer" (Ball 2002) are indeed superficially appealing, given the great success of modern computing theory and technology. Yet they lose their apparent content as we look at them more closely. If everything computes merely by virtue of existence, then what more do we say about an object when we call it a computer? How are novel and unconventional computing devices to be characterized, if computing occurs universally and intrinsically in physical objects?

We present here a more discriminating view, in which the use of a physical system to carry out an abstract process (a computation) depends on a number of specific properties that both the physical device (the computer) and physical process (it computing) must have (Section 6.3). Not all physical processes constitute computing: A key element in physical computation is the use of a physical system to manipulate the representation of abstractly encoded data in specific processes. In presenting our framework, *Abstraction/Representation (AR) theory* (Horsman et al. 2014; Horsman 2015), we show that physically carrying out computation and doing science are closely related activities (Section 6.2). Both involve *representational activity*. By looking at how computers are developed from both fundamental science and then engineering and technology, we show the crucial physical nature of computing.

Key to AR theory is the *representation relation* between physical objects and objects in the abstract mathematico-logical domain. AR theory takes as primary a realist physical domain; we discuss other potentially compatible views in Section 6.2.1. The representation relation is structured and directed, from physical to abstract: Scientific modeling is fundamental in AR theory. We consider representation functionally, in terms of its use and properties within the physical sciences. We are not primarily concerned with it as semantics or meaning, or, relatedly, as knowledge and information. AR theory's construction of computation is not of symbol manipulations, nor are there notions of representational states (Putnam 1960; Fodor 1975). If science is to happen in a physical external world, and any part of reality that corresponds to what is considered as an abstract world (where mathematics, logic, computations, etc. live), then there must be a map from the physical to that abstract world: Representation is that map.

AR theory separates out (i) physical systems and processes, (ii) abstract objects, and (iii) the representation relation that maps between them. Abstract entities do not mirror physical ones (Putnam 1960; Rorty 1979), but stand in quasi-functional relations to them. By analyzing the role and specific functionality of representation in science and technology, we give the AR framework for computing in terms of the interrelations of these three elements. In the specific context of computing, the representation relation allows abstract computations to be instantiated in physical computing systems, and the effects of physical processes to be represented as abstract computational results. There are not some physical states that are computational states; rather, there are computing *cycles* that require all elements to be present in specific ways before computing can be said to occur.

AR theory's analysis of physical computation captures computing in commuting-diagrammatic form: Physical computers use representation to act as predicting devices for the results of abstract computations. This acts as the converse situation to that of experimental science, where a physical theory functions as an abstract predictor for the behavior of physical systems. AR theory shows us the deep structural connections between computing and natural science. Experimental science and physical computing both require the interplay of abstract and physical objects. This is mediated via representation in such a way that the diagrams of AR theory commute: The same result is gained through either physical or abstract space-time evolution.

The central role of representation leads to the requirement for a *representational entity*. This is the entity that supports the representation relation between physical and abstract. For the natural sciences, this is the human experimenter or theorist. Within a computational process this is usually the programmer, designer, and/or end-user. However, AR theory does not require the representational entity to be human, or even conscious (Section 6.2.5).

AR theory treats both physical and abstract objects, mediated by representation, within the same framework. This allows us to discuss the connection between the scientific description of a system or device (as a theory of physics or biology, say) and its computational description as two distinct representations. We can use this distinction in the search for novel systems whose physical properties allow us to compute in new and interesting ways. Computer science has previously lacked a formal connection between physical device and abstract theory. AR theory now provides this, distinguishing the physical system from its abstract scientific and computational representations. Conflation of this three-way separation lies at the root of much of the confusion that surrounds both the development of unconventional computing and the relationship between scientific theory and computation (Section 6.4).

Not everything that supports a scientific representation supports a computational one. In this way, pancomputationalism is taken out of the picture by AR theory (Section 6.4.1). The connection between the physical system and an abstract representation is not in general coextensive: The physical system will have properties not captured in the abstract representation, and the abstract representation may have properties not realized in the physical system. Failing to take these differences into account gives rise to problems both of the existence of "side-channels" (Section 6.4.2) and the more extravagant claims of hypercomputing abilities (Section 6.4.3).

AR theory allows us to show the fundamental relationship between scientific and computational models, whilst preserving their necessary distinctiveness. Without this, a physical theory of computation tends to flounder. AR theory forms the backbone of a new framework for the foundations of computer science, treating this mutual relation between distinct models as fundamental: a natural science of computing.

6.2. Introduction to AR Theory

6.2.1 Science and Ontology

Key Role of Representation Natural science can be viewed, at its basic level, as concerning objects from two distinct domains. Objects, processes, and systems within the physical world are the subject of scientific theories scrutinized during experimental observations, and may be manipulated during experimental tests. Abstract objects are used to model these physical systems within the domain of mathematics, logic, or any other language of the sciences. The fundamental operation of science is this modeling of a physical object by an abstract one: the process of *representation* (Frigg 2006; van Fraassen 2008). This conception of science, and the crucial role of representation, is the starting point for the framework of AR theory.

Figure 6.1 illustrates a physical system represented as an abstract model. This is a fundamental use of the *representation relation* $\mathcal{R}_\mathcal{T}$, which quasi-functionally relates physical objects to abstract ones; it is not a mathematical function, as that would require both domain and range to be abstract. In Figure 6.1, a glass bead is represented using $\mathcal{R}_\mathcal{T}$ as a volume V. In general, we can talk about a domain **P** of physical objects and a domain M of abstract objects. (We use boldface for physical objects, italics for abstract ones,

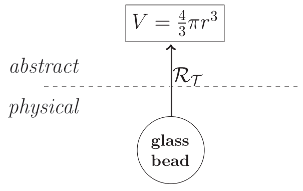


Figure 6.1 A spherical glass bead represented by its volume V. The representation relation $\mathcal{R}_\mathcal{T}$ relates the physical object to its abstract model, within the theory $\mathcal{T}$

and script for representations.) Representation is theory-dependent, which we denote by the subscript $\mathcal{T}$ in $\mathcal{R}_\mathcal{T}$. In this case, the theory is that of the classical mechanics of solid objects. If we were instead interested in the refractive properties of a glass sphere, we would represent it by its refractive index and surface geometry, and the theory would be classical optics.

There is no unique theory for a given physical system: Multiple theories can have different forms of validity, depending on the properties of the physical system in question, the degree to which they need to conform to experiment, and the domain for which they are required to be applicable.

Representation, Ontology, and Realism The field of study of representation in philosophy is vast. Since Rorty’s demolition of the notion of representation as a simple mirroring of abstract and physical (Rorty 1979), the nature and structure of different types of representation have been extensively explored. AR theory’s representation is primarily that of scientific representation: The physical world is fundamental, and representation is structure given abstractly on top of it. As used here, it does not come pre-loaded with implications of intentionality or meaning; it is, at its most fundamental, a mapping from physical to abstract. We almost never talk about “information” or “knowledge” or “meaning” in using AR theory. AR theory draws strongly on the framework for scientific representation explored by van Fraassen (2008), and with many similarities (as a more developed framework) to the basic modeling theory of Hughes (1997). For a philosophical overview of models in science, see, for example, Suppes (1960) and Frigg and Hartmann (2016).

Views of computation that regard it as the manipulation of representation have historically come from a semantic account of computation; see, for example Putnam (1960) and Fodor (1975). This comes out of an emphasis on the similarities between computation and cognition, thought, and language use. The abstract – thoughts, representation, language – is considered primary. The set of all objects is the set of abstractions: “The facts in logical space are the world” (Wittgenstein 1922, prop. 1.13). In such a view, if anything is to be called into question, then it is the existence of the physical world. At best this is idealism; at worst, hard-line logical positivism.

This concept of computation, as closely allied with cognition and a philosophy of logic and language, is the opposite of our starting-point for AR theory: a computation is abstract; a computer is physical. The question is not "what types of abstract states and processes form computations?" but now "which physical systems are computing?" We assume the existence of a physical world. This is the space of all physical objects, **P**. Idealist-leaning, nominalist, or verificationalist projects fail ultimately in their inability to abstract out ongoing and necessary interaction with an external world. Carnap's project for a basic sense-data language (Carnap 1950a) – which can be viewed in computer science terms as analogous to a universal concrete semantics– failed in ways that make it clear it cannot succeed. Following Quine (1971), our epistemology is naturalized: how abstract content interfaces with physical reality is found by interrogating our best scientific theories. In AR theory this also includes representation: Representation is always within the framework of a physical theory.

We do not use a cognitive starting-point for representation and computation in AR theory, but we do not reject outright the applicability of questions around intention and mental representation to notions of computing. The relationship between abstract computation and physical computer is the mind/body problem *du jour*, and questions of computation and cognition have a lot to say to each other. AR theory draws on many aspects of the long tradition of work around the abstract/physical divide concerning both mind/body and theory/reality in science, prioritizing the latter pair rather than the former.

Such a notion of "naturalized representation" gives the starting-point for AR theory: how representation functions within scientific theories. It is through scientific representation that we come to computing. The physical world and device are basic, and representation is structure in addition. This inversion of the usual conception of representation is shown in the direction of its basic mapping; $\mathcal{R}_{\mathcal{T}} : \mathbf{P} \to M$, rather than an abstract object representing a physical one, $M \to \mathbf{P}$.

AR theory starts from scientific realism, but need not end there. In terms of realism, the AR framework gives a way to talk about the physical world **P** without adding a structural commitment to one "foundational" or "fundamental" representation. It is not meant to solve all the issues of realist theories, but to give a structure compatible with a solution. The level of basic physical ontology in AR theory is very stripped down. The term "the physical world" can be viewed as a placeholder for whatever the physical world is, not identical with a representation of it. The aim here is to give a framework in which ontology and representation are considered separately, and the threads of multiple representations – physical, mathematical, computational, amongst others – can be teased apart. This avoids identifying fundamental ontology with whatever our favorite physical theory says it is this week. It also avoids having to choose one description as "more" fundamental than another (particles, fields, qubits, membranes, etc.). This removes the question-begging presumption of a relationship between different representations ("how can a collection of Fock spaces *be* a computer?") where there may be no such relationship. It also removes the issue in the foundations of computer science where a physical computer is frequently, and incorrectly, considered as identical with its concrete semantics.

By stripping the physical world of its representational structure, we are left with what might be termed the "fundamental problem of representation." **P** is supposed to be the set of physical objects without representational structure, yet to write down "**P**" is to violate this by giving it a representation. One response is to accept this necessary use of representation to perform communication as a Wittgensteinian ladder that is cheerfully kicked away after use. A less glib response is to take the notion of a naturalized representation and apply it to the whole system here: ontology is given by our best scientific theories, not just in what it is represented as, but in what is considered to exist physically.

This identification of that which is being represented as basic ontology necessarily opens us up to criticism by pathological case. For example, empty space is not presented in many physical theories, but becomes a thing-in-itself in quantum field theories. A frequent response to these issues of realism is scientific empiricism: what is being represented are events of observation. It may well be that AR theory is compatible with such a view. In this case then the domain **P** changes depending on context: It is, as it were, representational turtles all the way down. This is the empiricist way van Fraassen (2008) views scientific representation. It is also worth noting the fit between the commuting-diagrammatic structure of AR theory, below, and Quinean notions of the "field of force" at the edges of scientific theories where they make contact with empirical reality (Quine 1951); the reader is invited there to view "the physical world" as a convenient shorthand for empirical observations, if this better fits their basic metaphysics. Otherwise, we note that any sane realist theory contains a physical domain, and we continue to consider in AR theory how this interfaces with the abstract representations that we use for it.

6.2.2 Prediction in Science

With the context established, let us see where such a framework can take us. First, let us consider a physics experiment on a system $\mathbf{p}$ represented by abstract model $m_{\mathbf{p}}$. The system evolves under some dynamics $\mathbf{H}$ as $\mathbf{H}(\mathbf{p})$ to become $\mathbf{p}'$. We have a theory of this process $\mathbf{H}$; call it $C_{\mathcal{T}}$. Applying $C_{\mathcal{T}}$ to $m_{\mathbf{p}}$ we obtain $m'_{\mathbf{p}}$. We would like to test our theory by finding out whether $m'_{\mathbf{p}}$ corresponds to the result of the experiment, $\mathbf{p}'$. We cannot compare $m'_{\mathbf{p}}$ to $\mathbf{p}'$

directly; we first have to use the representation relation $\mathcal{R}_{\mathcal{T}}$ on $\mathbf{p}'$ to obtain $m_{\mathbf{p}'}$. This includes the process of measuring the outcome of the experiment. We can then ask, does $m'_{\mathbf{p}} = m_{\mathbf{p}'}$, and if not, what is the difference between them?

We do not need equality for $C_{\mathcal{T}}$ to be a good theory of the dynamics of $\mathbf{p}$. No theory is perfect, and we have limited precision for our experimental measurements. For $C_{\mathcal{T}}$ to be a good theory, we just need "close enough," say $|m'_{\mathbf{p}} - m| < \varepsilon$ for some suitable measure $|.|$ and suitably small ε. This process is illustrated in Figure 6.2, an example of an *ε-commuting diagram*, where theory and experiment agree to within some parameter ε.

Of course, this is a vastly simplified picture of the scientific process, in which many such diagrams interlock and underpin each other to build confidence in theories through many different and repeated experiments. We are most emphatically not claiming to have solved the philosophical questions of how science is done, only that something like this must be part of the story. All we need for our purposes here is that a "good theory," however established, can be described by a diagram like Figure 6.2. There are similarities between AR diagrams and others used for describing physical computation, especially those given by Ladyman et al. (2007), Maroney (private communication), and, in computer science, abstract interpretation (Cousot and Cousot 1977). However, AR representation (i) goes between physical and abstract, not functionally from abstract to abstract, (ii) is directed, and (iii) includes ε-closeness conditions.

Once we have a sufficiently good theory, we can use it to predict the behavior of physical systems. Figure 6.3(a) shows an ε-commuting diagram with both the physical system and the abstract model of its time evolution providing their outcome of the process. Figure 6.3(b) shows the use of the theory to *predict* the behavior of the physical system, without actually carrying out the physical process. From this, we can see that not only do experiments guide

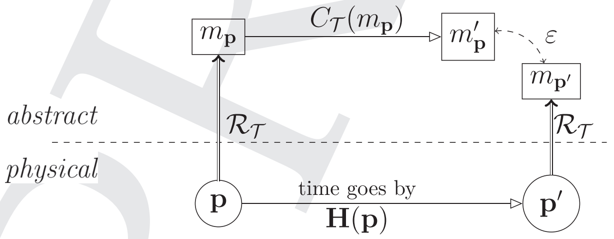


Figure 6.2 A simple experiment in which a physical system $\mathbf{p}$ evolves under some dynamics $\mathbf{H}$ to $\mathbf{p}'$. The corresponding abstract representations $m_{\mathbf{p}}$ and $m_{\mathbf{p}'}$ are obtained using $\mathcal{R}_{\mathcal{T}}$. The theory corresponding to the experiment $C_{\mathcal{T}}$ is used to calculate the expected outcome $m'_{\mathbf{p}}$, which is then compared with $m_{\mathbf{p}'}$, with resulting difference ε

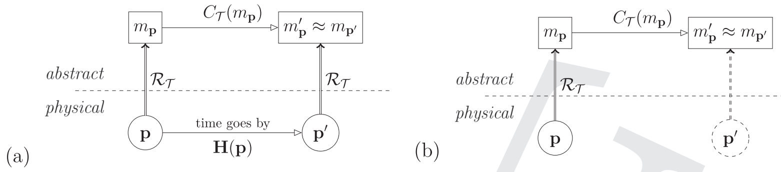

$m_{\mathbf{p}}$
$C_{\mathcal{T}}(m_{\mathbf{p}})$
$m'_{\mathbf{p}} \approx m_{\mathbf{p}'}$
abstract
physical
$\mathcal{R}_{\mathcal{T}}$
$\mathbf{p}$
time goes by
$\mathbf{H}(\mathbf{p})$
$\mathbf{p}'$
(a)
(b)

Figure 6.3 (a) A “good theory” has $m'_{\mathbf{p}} \approx m_{\mathbf{p}'}$; (b) this allows the outcome of a physical process to be *predicted*, without having to check by running the experiment

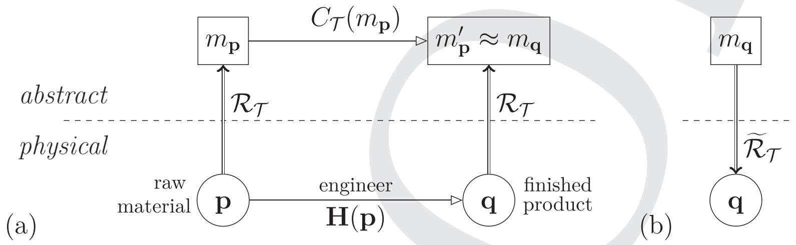

$m_{\mathbf{p}}$
$C_{\mathcal{T}}(m_{\mathbf{p}})$
$m'_{\mathbf{p}} \approx m_{\mathbf{q}}$
$m_{\mathbf{q}}$
abstract
physical
$\mathcal{R}_{\mathcal{T}}$
$\widetilde{\mathcal{R}}_{\mathcal{T}}$
raw material
$\mathbf{p}$
engineer
$\mathbf{H}(\mathbf{p})$
$\mathbf{q}$
finished product
(a)
(b)

Figure 6.4 (a) The engineering process: making a $\mathbf{q}$ from a $\mathbf{p}$ using process $\mathbf{H}$ with theory $C_{\mathcal{T}}$; (b) the instantiation relation $\widetilde{\mathcal{R}}_{\mathcal{T}}$ is a shorthand for the engineering process

the development of theories to explain their results, but theories can predict the outcome of experiments that have not yet been carried out, suggesting directions for new experiments. Both of these are observed as part of scientific practice. Moreover, good theories are not only used to guide further scientific research, they are also used to underpin new technology.

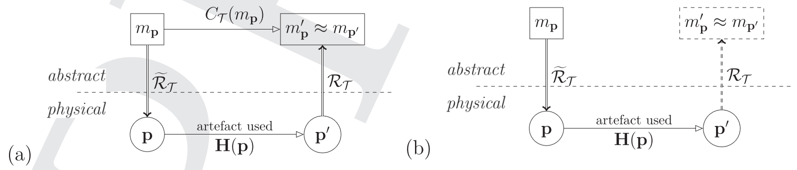
6.2.3 Technology and Engineering

Instantiation: Engineering an Artifact A good theory as illustrated in Figure 6.3 tells our the outcome of $\mathbf{H}(\mathbf{p})$ without our having to do the experiment. Furthermore, it tells us that we can reliably make a system $\mathbf{p}'$ from a $\mathbf{p}$ using a process $\mathbf{H}(\mathbf{p})$ that we understand well through the theory $C_{\mathcal{T}}$. This is *engineering*; again, we are not claiming to have solved the philosophical questions around engineering. Our ability to make things from detailed designs is unquestionable, evidenced by the multitude of high-tech gadgets available for purchase in their millions. What is most relevant to the development of our framework is that engineering effectively reverses the representation arrow $\mathcal{R}_{\mathcal{T}}$, allowing us to instantiate theoretical objects in certain well-defined circumstances. This is illustrated in Figure 6.4(a), in which a product $\mathbf{q}$ is made from raw material $\mathbf{p}$. The theory provides the method for the engineering process $\mathbf{H}(\mathbf{p})$, and the abstract comparison $m'_{\mathbf{p}} \simeq m_{\mathbf{q}}$ verifies that the finished product $\mathbf{q}$ is sufficiently close to the theoretical specification. We can abbreviate this by the instantiation relation $\widetilde{\mathcal{R}}_{\mathcal{T}}$, Figure 6.4(b), in which the abstract model $m_{\mathbf{q}}$ is instantiated as a physical object $\mathbf{q}$. It is important to note that representation and instantiation are not symmetric processes: making models that represent physical systems is easier than making physical objects that instantiate abstract models. In particular, it is possible to devise *unphysical* abstract models that have no possible real-world physical instantiation.

Using an Engineered Physical Artifact Given a “good theory,” we can use it to *engineer* systems, in concert with instantiation as described above, and then put them to use. For example, we probably want to test that our artifact does in fact conform to the engineering specification for its intended use. Figure 6.5 illustrates this process.

Figure 6.5(a) shows the ε-commuting diagram with both use and theory providing their outcomes of the process, allowing its suitability for the task to be checked. Figure 6.5(b) shows the artifact used to *predict* the outcome of the theory without actually carrying out the abstract calculations. This is what normal use of an engineered artifact corresponds to: Our confidence in the theory behind the engineering allows us to use the artifact without having to check it will do what we want it to do.

While the full diagram for engineering in Figure 6.5(a) looks superficially similar to the full diagram for science in Figure 6.3(a), there are fundamental distinctions. The first difference is the starting point of the process: note the instantiation arrow in Figure 6.5(a). For science, the starting point is a physical system to be modeled (represented) and understood. For engineering, the starting point is a problem encoded into an abstract model (engineering specification), to be engineered (instantiated) as the desired physical artifact. Science starts with the *physical* systems, engineering starts with the *abstract* models. The second difference is in the desired endpoint, or goal, of the scientific or



$m_{\mathbf{p}}$
$C_{\mathcal{T}}(m_{\mathbf{p}})$
$m'_{\mathbf{p}} \approx m_{\mathbf{p}'}$
abstract
physical
$\widetilde{\mathcal{R}}_{\mathcal{T}}$
$\mathcal{R}_{\mathcal{T}}$
$\mathbf{p}$
artefact used
$\mathbf{H}(\mathbf{p})$
$\mathbf{p}'$
(a)
(b)

Figure 6.5 (a) A well-engineered system has $m'_{\mathbf{p}} \approx m_{\mathbf{p}'}$; (b) this allows the outcome of an abstract calculation (of the designed behavior of the physical system) to be *predicted* without having to check by doing the abstract design calculation engineering process. For science, the goal is a theory that describes reality sufficiently well. For engineering, the goal is a physical artifact that meets the specified design. Scientific goals are abstract, engineering goals are physical.

As a consequence of these different goals, the response to an insufficiently small ε is different in science and engineering (once the possibility of a faulty experiment or incorrect specification has been eliminated). In science, when ε is too large, it means that the theory fails to adequately describe physical reality, and so the theory needs to be improved. In engineering, when ε is too large, it means that the engineered product fails to meet the theoretical specification, and so the physical object needs to be improved.

6.2.4 *Computing Technology*

Abstract Prediction Computers are one type of physical system among the many and varied things that we engineer. However, they differ in one fundamental way from the engineered artifacts described above. For computing, the goal is to carry out an abstract computation that is (in general) unrelated to the details of the physical computer. We can now use our framework to address our original question about when a physical system computes. Figure 6.6 shows a physical system $\mathbf{p}$ carrying out an abstract computation $C_{\mathcal{T}}$. Note that we are not here addressing what explicitly characterizes this abstract *computation* (for which see, for example, Piccinini [2015, 2017] on semantic accounts of physical computation); rather, we are addressing the question of when we can say that a physical system is *computing* an abstract *computation*.

In Figure 6.6, $m_{\mathbf{p}}$ is the encoding of our problem into a suitable abstract computational model; we discuss the encoding stage in more detail shortly.

Given this abstractly encoded problem $m_{\mathbf{p}}$, we instantiate it in the physical computer $\mathbf{p}$ and let it run. If successful, the result $\mathbf{p}'$ can be inspected to obtain the abstract answer.

While the full diagram for using an engineered artifact in Figure 6.5(a) is identical to the full diagram for computation in Figure 6.6(a), there is again a

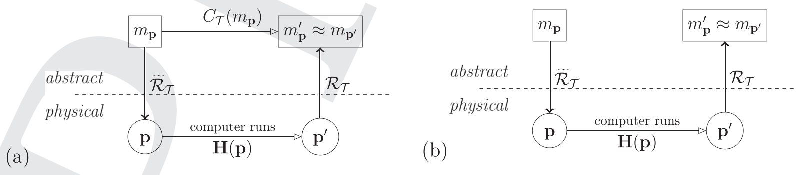

(a)
$C_{\mathcal{T}}(m_{\mathbf{p}})$
$m_{\mathbf{p}}$
$m'_{\mathbf{p}} \approx m_{\mathbf{p}'}$
abstract
physical
$\widetilde{\mathcal{R}}_{\mathcal{T}}$
$\mathcal{R}_{\mathcal{T}}$
$\mathbf{p}$
$\mathbf{p}'$
computer runs
$\mathbf{H}(\mathbf{p})$
(b)
$m_{\mathbf{p}}$
$m'_{\mathbf{p}} \approx m_{\mathbf{p}'}$
abstract
physical
$\widetilde{\mathcal{R}}_{\mathcal{T}}$
$\mathcal{R}_{\mathcal{T}}$
$\mathbf{p}$
$\mathbf{p}'$
computer runs
$\mathbf{H}(\mathbf{p})$

Figure 6.6 (a) A well-engineered instantiated computer has $m'_{\mathbf{p}} \approx m_{\mathbf{p}'}$ for computations it is capable of performing; (b) this allows the outcome of an abstract computation (the desired goal of computation) to be *predicted*

Table 6.1 *The essential differences between science, engineering, and computation: start: whether the starting point is a physical system* **p***, or an abstract specification* $m_{\mathbf{p}}$*; goal: whether the desired goal is a physical system* **p***, or an abstract result* $m_{\mathbf{p}}$*;* ε *too large: what part must be changed to make* ε *sufficiently small*

	start	goal	ε too large
science:	physical	abstract	change $m_{\mathbf{p}}$
engineering:	abstract	physical	change **p**
computation:	abstract	abstract	change **p**

fundamental difference, and again this is in what is considered the goal of the process. For engineering, the goal is a physical object that meets the specified design. For computing, the goal is an abstract result of a computation. Engineering results are physical, while computational results are abstract. These differences are summarized in Table 6.1.

Performing a computation is a form of engineering where the desired result is an abstract representation rather than the physical system itself.

Encoding and Decoding Our initial discussion of physical computation above skips over some important details. Unlike the science and engineering diagrams so far, where the physical system **p** has been directly represented by the model $m_{\mathbf{p}}$, we now have an abstract calculation that is initially unrelated to the physical computer or our model of the computer. The calculation problem may be the reason that the computer has been engineered in the first place (as with the earliest computers, or with modern specific-use devices), or it may be a new problem that the user has reason to believe is amenable to being solved on existing hardware. In either case, there is though no a priori connection between the abstract specification of the problem, c, and the abstract specification of the computer, $m_{\mathbf{p}}$. This connection is to be found in the process of encoding. Figure 6.7 shows a physical computer **p** being used to do calculation c. Since c is in general unrelated to the computer we want to use, the first step is to map the calculation onto the model of the computer $m_{\mathbf{p}}$. This *encoding* step includes checking that the computer is capable of representing the calculation (i.e., has enough memory and a suitable set of operations).

A modern programmable computer has a lot of existing programs (software, apps) to assist with the process of encoding a new problem, each of them running their own computations, resulting in many nested computational processes. It is easier to see how the basic encoding process works on a simpler computer, such as a pocket calculator. Suppose we have a restaurant bill for £93.47 that we need to divide equally between seven guests (thereby ignoring

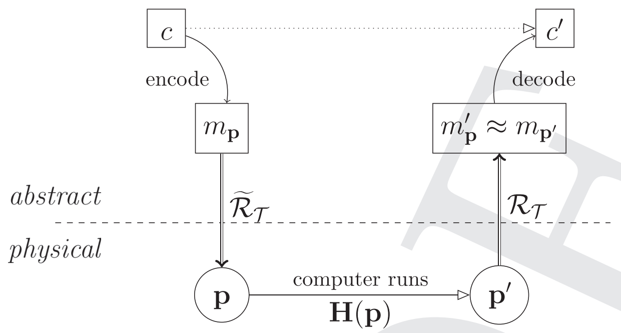


Figure 6.7 Using physical system **p** to carry out the abstract computation *c*

complications of who had the rice). The sequence of button presses `9 3 . 4 7 ÷ 7 =` will, on most calculators, result in it displaying the answer. That is an almost trivial encoding where the calculation can be entered straight into the calculator.

In general the result will need to be *decoded* from the abstract representation $m_{\mathbf{p}'}$ of the final computer state $\mathbf{p}'$. In the case of the restaurant bill, the decoding is again straightforward. Reading the display you announce that each person owes £13.36. The decoding you do is to interpret the array of seven segment displays as a number, then add the "£" to that number to interpret it as an amount of money. And you also rounded the amount up to a full penny, because amounts of money smaller than this are not useful for settling a restaurant bill.

Encoding and decoding are important steps in physical computation. Indeed, one of the characteristics of the associated representational processes is that there is an element of choice about the encoding. The possibility of different choices implies both that the same abstract computation can be carried out on different physical computers, and that the same physical computer can in general carry out different computations. What is important is that the computational semantics of the operations is given by the information that is processed by them, and that there are different choices of the physical process by which that happens. The possibility of different choices is one of the ways by which we can see that computation is indeed happening.

6.2.5 Representational Entities for Computation

AR theory enables us to specify the elements of representation and the interplay of physical and abstract that happen during the use of computers by human beings. It is not, however, restricted to this. A common, if unsatisfactory, counter to pancomputationalism is to restrict the definition of a computing system to one used by conscious users via a definition of information (processing) that is strongly semantic and intentional; see, for example, Bar-Hillel and Carnap (1953) and Mackay (1969).

AR theory does not do this. Within AR theory, the requirement is not for a human being, or even an entity that can think or communicate; the requirement is for representation. We thus do not need to take a position in the ongoing discussions of whether representation requires a thinking or human entity, and we refer to the compute cycle requirement as a *representational entity*. As we discuss in Section 6.2.1, we take a realist view: Whatever the ultimate ontology of "abstract" objects is, there is only ever access to represented objects through physical systems capable of supporting representation. The paradigm example is the human brain: We use representation to model physical objects in the world around us (including ourselves) as abstract notions. This representation happens, though, in the physical brain, for example, when a human uses a laptop computer. In such a case, we are the physical entities using the representation relation: We are the representational entities.

Thus, we have two conceptually distinct, but not necessarily physically distinct, physical entities. First, there is the physical object **p**, as labeled in the diagrams in Section 6.2, that participates in representational activity (be it science, engineering, computing). Second, there is the representational entity **e** (denoted with bold font as, *ex hypothesi*, the representational entity must be physical) that supports the representation relation $\mathcal{R}_\mathcal{T}$ it is using for **p**. We say that the system comprising **p**, **e**, and $\mathcal{R}_\mathcal{T}$ forms a *closed representational system*. If the cycle is a compute cycle, then this system forms a *closed computational system*: the *system* is computing.

In AR theory, the locatedness of the representational entity is important for determining the type of representational activity happening in a system. If the system comprising **p** and $\mathcal{R}_\mathcal{T}$ does *not* include the physical representational entity **e**, that is, if **p** and **e** are physically as well as conceptually distinct, then we say that the system is *open under representation*. In all the examples of human-designed computer use given above, the steps that go across the divide between physical and abstract (the representational and instantiational steps) all rely on a human representational entity. This entity is separate from the system that is the computer. So in human-designed computing, the computer alone (laptop, Difference Engine, slime mold, etc.) does not form a closed representational system: The representational entity is separate from the computing device, and not even necessarily co-located with it. Human-designed computers are open under representation, and require a human representational entity to close them.

Identifying the steps in a computation (or other representational activity) will identify the representational entity. Horsman et al. (in press) argue that it is possible to have a closed computational system without a human representational entity, analyzing the example of a bacterium performing chemotaxis, that is, changing its direction of motion towards a source of food. In such a closed computational system there are additional challenges to identifying the computational steps: a non-human representational entity cannot communicate to us that it is using representation. It therefore falls to us to determine if the system under consideration is itself using representation. That is, can we represent it as a closed representational system? We do not assign ourselves as the representational entities here: Rather, we determine whether we can describe the entire system as using the representational aspect of parts of itself in certain processes. A physical system that becomes representational only when an $\mathcal{R}_\mathcal{T}$ is given by a human observer is *not* a closed representational system.

Thus the AR framework is able to discriminate between computing behavior and non-computing processes even in the absence of intelligent users or designers. The representation relation is always used by some entity, though, and it is that entity which is using the interfacing between abstract and physical that is a key part of physical computation.

6.3. Identifying Computing with the AR Framework

With the AR framework on board, we can use it to generate criteria that distinguish "computers" from other elements of physical reality. One of our original motivations for developing the theory is to provide a critical evaluation of proposed unconventional computational devices. Here we apply it to slime mold computation, which has received significant attention in the past decade. First, we show how the framework describes our familiar digital computers, and also one of the earliest computers, the Babbage Difference Engine. By identifying the components that make up processes that are known to be computing, we can then extend this into the territory of novel and unconventional computing.

Figure 6.7 shows the six essential components to an AR framework description of a computation: **theory**, **encoding**, **instantiation**, **physical process**, **representation**, and **decoding**. For each example, we first list what each of those components consist of, then discuss any issues that arise in identifying the components.

6.3.1 Classical Digital Computing

By classical digital computing, we mean the technology that underpins the computers we use on our desks, as laptops, in our smartphones, running the internet, and in many other types of technology, such as modern cars. We take this to be computing, uncontroversially. By showing how the components fit together to produce a compute cycle in this technology that is nowadays the paradigm example of computing, we pave the way for demonstrating it in nonstandard devices.

Theory: The theory of classical computing covers the hardware (including how the transistors implement Boolean logic, and how the architecture implements the von Neumann model) and the software (including programming language semantics, refinement, compilers, testing, and debugging).
Encode: The problem is encoded as a computational problem by making design decisions and casting it in an appropriate formal representation.
Instantiate: Instantiation covers the hardware (building the physical computer) and the software (downloading the program and instantiating it with input data).
Run: The program executes on the physical hardware: The laws of physics describe how the transistors, exquisitely arranged as processing units and memory, and instantiated into a particular initial state, act to produce the system’s final state when execution halts.
Represent: The final state of the physical system is represented as the abstract result, for example, as the relevant numbers or characters.
Decode: The represented computational result is decoded into the problem’s answer.

Despite the complexity of today’s computers, the underlying theory is highly developed and well-understood, the result of years of development and testing as each advance in functionality is introduced.

6.3.2 Babbage’s Difference Engine

There is ongoing debate about the first “true” human-designed computer (with Stonehenge and the Antikythera mechanism, amongst others, vying for the title). The Babbage Difference Engine is one of the first recognizably modern computing devices. In particular, it was the forerunner of the Analytical Engine, the first proposed programmable machine with associated programming, given by Lovelace (1843) as she laid the foundations of modern computer science. The Difference Engine was, unlike the Analytical Engine, actually built. It computes tables of logarithmic functions by approximating them as sums of polynomials. It therefore needs to find the value of a function, e.g., $f(x) = x - \frac{1}{2}x^2$, for a whole set of values $x = 1, 2, 3, \ldots$ It uses the “method of differences” to change the problem into one of addition and subtraction, finding first the difference between subsequent function values (the “first difference”), and then the difference between subsequent values of the first difference (the “second difference”), and so on, a degree-n polynomial having n differences. Essentially it mechanizes a difficult calculation (a logarithm) by turning it into a relatively straightforward one (first into polynomials and then into addition). The addition is performed physically by combining the rotations from different cogs in the device, each of which is set to the required input.

Theory: The theory of the Difference Engine as a device comprises the theory of how a set of interacting gears can generate an addition function. Crudely, this can be thought of as combining the rotation of separate cogs (the inputs) onto other cogs (the output). The Difference Engine theory also includes how the addition operations for each difference combine to form the correct addition function to generate the next value of $f(x)$.
Encode: The problem (calculate a logarithm) is encoded as an approximation of sums of polynomial functions, which are in turn reduced to addition functions.
Instantiate: The computation is instantiated firstly in the engineering of the device itself: the hardware is not programmable. The input is then given by turning the input dials to the settings that correspond to the first n values (usually around four) of the function $f(x)$ to be determined.
Run: The Engine runs (powered by turning a crank handle). As the gears interact, differences are calculated and then added.
Represent: The final position of the output gears is coupled to a “printer,” whose written numerical output depends on the position of the cogs.
Decode: The value of the $f(x)$ for the specific x computed by the Engine is used to calculate the required logarithm (by hand).

The Difference Engine demonstrates how computation can occur without full programmability. The Engine can perform only specific addition tasks, given by its construction. Addition is “hard-coded” into the design of the physical device. The Engine performs a range of calculations by virtue of the fact that it can take different inputs. This is one way we can see that the device is processing information through its physical operation. The theory of the device was originally developed to design clockwork. However, small differences between the physical system and the model originally made the operation faulty, as errors cascaded through the system. While these are resolvable with modern precision-engineering (for example, the Difference Engine in the London Science Museum works with negligible errors), the much more sensitive Analytic Engine has still not been constructed with a physical gearing that matches the necessary precision of its theory.

6.3.3 Slime Mold Maze Solver

Adamatzky (2010) documents a range of computational uses for the slime mold *Physarum polycephalum*. Nakagaki et al. (2000) describe the observation of the slime mold finding the shortest path through a maze. That work is described in experimental terms: testing a theory of slime mold behavior in the presence of food.

The maze to be solved is implemented as a small physical structure, a few centimeters across, suitable for a slime mold to inhabit (Figure 6.8). The tested maze has multiple possible routes and several dead ends. Initially, the slime mold is grown to fill the whole of the maze. Then blocks of slime mold food, oat flakes in agar, are placed at the entrance and exit of the maze. The behavior of the slime mold is then observed over the next few hours. Having discovered the food, it withdraws from the dead ends of the maze and concentrates along the shortest path(s) between the two food blocks. After four hours, the slime mold has withdrawn from all the maze dead ends, but still exists along parallel alternative paths. After eight hours, the slime mold has withdrawn from the longer parallel routes, and has found the shortest path.

Now having evidence for that theory, we can exploit the same process to *compute* the shortest path.

Theory: Slime mold forms a minimum-length body between food sources, as a consequence of the way its contraction frequency changes in the presence of food (for which see references cited in Nakagaki et al. [2000]).
Encode: If the abstract problem is to compute the shortest path through the maze, or simply any path through the maze, the encoding is essentially trivial: $c = m_{\mathbf{p}}$. If the maze abstraction is a more indirect analog of some other problem, the encoding would be more complex. Analogs tend to be fairly direct encodings, exploiting a clear analogy.

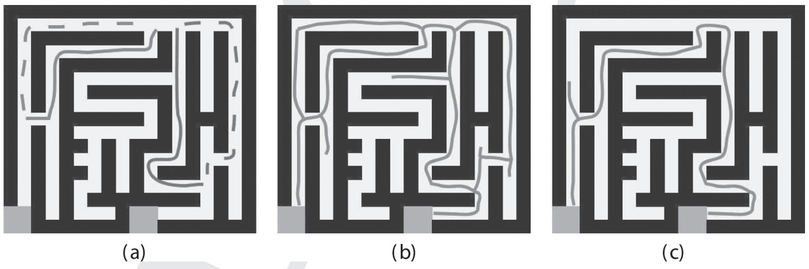


Figure 6.8 The slime mold *Physarum polycephalum* physically computing a path through a maze: diagram after Nakagaki et al. (2000). The dark areas are the maze walls; the pale regions are the maze. The gray squares indicate the position of the oat-flake-containing agar food blocks at the entrance and exit of the maze. (a) Multiple paths: on the left (“α routes”) the solid path fragment is 20 percent shorter than the dashed path; on the right (“β routes”), the solid path is 2 percent shorter than the dashed path. At the initial state of the physical system $\mathbf{p}$, the maze is filled with slime mold; (b) half way through execution, after four hours: the slime mold (marked as a line) has shrunk and moved out of all the maze dead ends; (c) the final state of the system $\mathbf{p}'$: after eight hours, the slime mold has found the shortest path

Instantiate: This has three main parts: build a physical maze that instantiates the abstract maze problem from materials supporting slime mold reconfiguration; place food sources (oat flakes) at the entry and exit positions; cover the maze with pieces of slime mold.
Run: Allow the slime mold pieces to coalesce into a single organism, and then wait for it to contract to the shortest path through the maze. The reported system took approximately eight hours to run, on a maze of approximately 4×4 cm.
Represent: Read off the final position of the slime mold in the maze, which requires the use of image processing to detect, and represent this as the position in the abstract maze.
Decode: Decode the abstract slime mold position into a route through the maze.

In this example of unconventional computation, we can see how the AR framework allows us to analyze the claims of computation.

There is first an issue at the level of the theory: The minimization is approximate. Nakagaki et al. (2000) report results from 19 experiments. In two cases no path formed. In three cases the slime mold did not fully contract, occupying all branches. When it did contract to a single route, for the β routes differing by only 2 percent, it chose the shorter route five times and the longer route six times. When it contracted to a single α route, which differ by 20 percent, it always chose the shorter route. So the theory would be better stated that the slime mold contracts to approximately the shortest path, most of the time, for mazes of this size. Hence, to use this system as a computer, we have to be willing to accept a quite large ε, and run the computation several times. Additionally, there is no evidence that the approach can be scaled to large mazes. It requires further scientific experiments to determine the domain of applicability and the degree of approximation. Note also the potentially considerable computation required in the form of image processing during the representation stage for detecting the position of the slime mold within the maze. Such additional computation needs to be considered when calculating the computational power of a physical device.

In this slime mold example, there is no clear distinction between computer (hardware), program (software) and input data (configured run of software). Moreover, the construction and programming effort can no longer be amortized over a potentially unbounded number of runs. Zauner and Conrad (1996) argue that one-shot "instance machines," which can solve only a single instance of a problem, have their advantages for certain substrates. Such machines avoid the need for resetting to some initial configuration, and so the compute step can irreversibly alter the state of the physical system, which is often a necessity when using a complex biological substrate. However, the cost of each instantiation needs to be low for this to be a viable strategy, significantly lower than the cost of a reusable computer for the same problem.

6.4. Distinguishing Representation(s) and Reality

We have seen in Section 6.3 how AR theory allows us to analyze specific systems for their computational activities. Separating out physical system, physical theory, and computational representations enables us to identify when all these elements, and the necessary connections between them, are present. This separation, and AR theory in general, can also be used to help design unconventional computer architectures, and to address some issues and claims about computation.

Both the abstract model $m_\mathbf{p}$ and the physical system $\mathbf{p}$ are essential components of AR theory, as are the instantiation and representation relations between them. Failing to properly distinguish which claims are about the abstract model and which are about the physical system can cause confusion. Confusion also arises by failing to differentiate claims about different abstract representations.

6.4.1 Pancomputational Rocks Aren't

Pancomputationalism is the view that everything – rocks, hurricanes, planetary systems, galaxies – are computing systems (Piccinini 2017).

The weak form of pancomputationalism holds that every system is computing (at least) itself. Consider the claim of a rock computing "itself" in the framework of AR theory, Figure 6.6. The rock is the physical system $\mathbf{p}$. But in this context there is, importantly, no representational entity $\mathbf{e}$ and hence no representation relation $\mathcal{R}_\mathcal{T}$. So there is no encoding of a computational problem, no relevant abstract model $m_\mathbf{p}$ representing $\mathbf{p}$, no instantiation of that model as the rock, no representation of the rock's final state back to an abstract result, and no decoding of that abstract result into the solution of the problem. There is only the physical rock $\mathbf{p}$, which does not form a *closed computational system* (Section 6.2.5).

One might try to argue that the rock is its *own* representational entity, that $\mathbf{p} = \mathbf{e}$. As we argue elsewhere (Horsman et al. in press), in AR theory such a claim requires us to demonstrate representational activity occurring. As we show in that paper, this is a highly non-trivial process requiring the active and explicit use of representation intrinsic to a system's processes. Even for organisms such as bacteria it is controversial to claim representational activity. It is highly implausible that the criteria can be established for rocks. In the AR theory definition, systems do *not* compute themselves *for* themselves, because they do not represent themselves.

The strong form of pancomputationalism holds that every physical system performs a combinatorially vast number of computations, of every finite automaton that its microstate can encode through some tabular representation (Putnam 1988). Again, consider the claim of a rock computing one of these automata in the framework of AR theory, Figure 6.6. Again, the rock is the physical system **p**. Now there is a representational entity **e**: a person pointing at the rock, allegedly encoding their problem, and decoding the result, by using a representation of the rock's relevant microstate, establishing the relevant table defining the automaton.

Despite the existence of all the components in the theory in this case, there is nevertheless an issue. All the "computation" of establishing the mapping from rock states to table entries is being done in the representation stage: the rock *itself* has computed none of this. The representational entity could equally well have pointed to *any* rock; a different representational mapping would be needed, and would need to be computed in its entirety without the aid of the indicated rock. One might equally say that a broken clock is measuring the time: we observe the final state of the broken clock, but then we must use another clock to establish the correct representation of its broken state: *All* the measurement of the amount of time that has passed is being done in the representation (using a second clock); the broken clock has measured none of this. Again, the identified components **p**, **e**, and $\mathcal{R}_{\mathcal{T}}$ do not form a *closed computational system*: A further computer is needed, and performs the totality of the purported computation. This contrasts with the case of a clock that is known to be, say, five minutes slow. Its physical state can be represented abstractly by **e** as one that is five minutes later in time than the standard representation would suggest. Alternatively, **e** could use the standard representation, and then decode the resulting abstract state by adding the five minutes. Here **e** is responsible for only a small, and well-determined, amount of computation to perform the representation and decoding.

So according to AR theory, there is no pancomputationalism, either weak or strong. For computation to occur at all, we require a representational entity, instantiation, and representation, in addition to the physical processes. For these identified representational entity, physical system, and representation relation to be sufficient to be performing the purported computation, they must additionally form a closed computational system. Rocks don't.

6.4.2 The Representation is Less than Physical Reality: Side Channels

The abstract model is just that: a model. It necessarily omits details about the physical system it is modeling, and may make simplifying assumptions. Other models, and their corresponding representations, are possible. Mathematical proofs are performed at the level of the model, and so concern only things in the model. If the physical system is richer, it can exhibit behavior not represented by the model, or the proofs. In particular, if a different model is used for a given physical system, with its own representation, different properties may hold than in the original model.

For example, a particular system, such as a crypto system, may be proved secure. Such a proof is performed at the abstract level, and may depend on assumptions about the physical system, particularly assumptions about what can be *observed* (and so represented) about the physical system. Different models support different observations; hence these may break the assumptions underlying the mathematical proofs.

Such alternative observations in the case of security systems, for example timing observations (Kocher 1996), are called *side channels*, and many kinds exist (Clark et al. 2005). The original analysis of kinds of side-channel in Clark et al. (2005) was performed purely in the context of abstract-level refinement concepts, although physical issues were considered. AR theory can augment such analyses by exploiting its clear distinction between physical-to-abstract representation relations and abstract-to-abstract mathematical refinement relations, helping to expose where properties can be subverted and attacked.

6.4.3 The Representation is More Than Physical Reality: Hypercomputation

The AR theory diagram only "ε-commutes"; there may be small differences between the desired computational result and the physically computed result. Digital systems are designed to commute exactly: Because a physical gap is engineered between the instantiation of abstract 0 and 1, small errors in the physical system either do not lead to errors in the representation and decoding, or can be identified and corrected. In contrast, continuous analog systems are not exact and have no such gap, hence errors can propagate.

When deducing properties of a computational system, it is important to realize that the abstract model can have different properties from the physical system. In particular, abstract models of "continuous" systems often model state variables using real numbers. This does not mean that the physical system somehow "implements" such real numbers. That this is so can be readily seen in some cases. For example, Lotka-Volterra-style predator-prey models (Wangersky 1978) use a real-valued variable to model the population size using a continuum approximation. The population size is in reality a discrete quantity, and such models break down when the continuum approximation is no longer valid. For another example, the Banach-Tarski paradox (Wagon 1985; Wapner 2005) is a theorem that states that it is possible to take a sphere, partition it into a finite number of pieces, and reassemble those pieces into two spheres each the same size as the original; the proof relies on properties of the reals that cannot be exploited to double a physical ball of material made of discrete atoms.

An (abstract) real number potentially has infinite information content. Claims of hypercomputing (systems that can compute some non-Turing computable functions) and super-Turing computing (systems that can compute some Turing-computable functions with exponential speedup over Turing Machines) that rely on this infinite information content being physically accessible appear to be confusing the mathematical real number power of the abstract model with the physical capabilities of the modeled physical system (Broersma et al. in press). If that abstract content could be exploited physically, it would lead to these claimed forms of hypercomputing and super-Turing computing. However, there is no evidence that physical systems can do this: infinite-precision real-valued variables cannot be *instantiated* in a physical material system. Physical variables such as position and momentum are classically *modeled* using real values, yet according to our current best physical theories, in particular quantum theory, the physical world is ultimately discrete, and so the set of values these variables can take is ultimately countable, and do not form a continuum. The real numbers used in the abstract model are just that: a model.

AR theory, with its careful distinction between the physical system and its abstract model, helps us to analyze the claimed computational power of various systems, determining whether the power is part of the abstract model, the physical device, or the representation relation.

6.5. Summary

Landauer (1996) famously claimed that "information is physical." The physical nature of computing has been acknowledged by the unconventional computing community, but computer science in general has hitherto viewed its subject matter to be one of mathematics and logic, relegating the physical details of computing devices to engineering. As various forms of non-standard computing come to prominence, in particular quantum and Internet-based technologies, this division has become increasingly untenable. Various fields from physics to biology to the social sciences have begun to import the language of information processing to describe their model systems in ways that are at odds with the usual foundations of computer science. AR theory allows us to bridge this divide, and to put the physical nature of computing devices back into the core of computer science, while also preserving its specific domain of applicability. Demonstrating the foundational part physical devices play in computing is not to extend the definition of "computer" to encompass every physical object. Computing is physical, but not everything that is physical computes.

The natural scientific description of a physical system and its computational description share many important properties. Both are model representations of the underlying physical system. They can relate to each other, and to the system being modeled, in a number of different ways. Carefully distinguishing these allow us to find when a physical system can support a computational representation, and when it is, in fact, being used as a computer. This is a complex process: A number of important criteria must be met, with, fundamentally, representation occurring and being used in specific ways in the physical system. Differences between physical system, physics-based models, and computational representations gives rise to many of the problematic behaviors of computing systems: side-channel attacks, over-ambitious claims of computing power, and lack of clarity about what in a computer is computing and when. AR theory gives a framework in which all these claims can be defined and then analyzed. This opens the field for a formal computer science of unconventional devices, and for a new foundational understanding of the relationship between computation and the physical sciences.

Acknowledgments

DH and VK are supported by EPSRC under Grant EP/L022303/1. SS acknowledges partial funding by the EU FP7 FET Coordination Activity TRUCE (Training and Research in Unconventional Computation in Europe), project reference number 318235. We thank the various referees for their detailed comments, which have helped improve this chapter.