European Journal for Philosophy of Science (2019) 9: 19
https://doi.org/10.1007/s13194-018-0237-1

PAPER IN GENERAL PHILOSOPHY OF SCIENCE

A refinement to the general mechanistic account

Eric Nelson Hatleback[1] · Jonathan M. Spring[1,2]

Received: 19 January 2018 / Accepted: 6 November 2018 / Published online: 7 January 2019

Abstract

Phyllis Illari and Jon Williamson propose a formulation for a general mechanistic account, the purpose of which is to capture the similarities across mechanistic accounts in the sciences. Illari and Williamson extract insight from mechanisms in astrophysics—which are notably different from the typical biological mechanisms discussed in the literature on mechanisms—to show how their general mechanistic account accommodates mechanisms across various sciences. We present argumentation that demonstrates why an amendment is necessary to the ontology (entities and activities) referred to by the general mechanistic account provided by Illari and Williamson. The amendment is required due to the variability of some components in computing mechanisms: the very same component serves as either entity or activity, both between levels and within the same level of the explanatory hierarchy. We argue that the proper ontological account of these mechanistic components involves disambiguation via explicitly indexing them as entities or activities.

Keywords Mechanisms · Computing · General mechanistic account · Ontology of mechanisms · Philosophy of computer science

1 Introduction

In synthesizing approaches in the mechanisms literature, Phyllis Illari and Jon Williamson establish the following claim: "A mechanism for a phenomenon consists of entities and activities organized in such a way that they are responsible for the phenomenon."[1] Their position is a carefully constructed synthesis of the three most respected mechanistic accounts.[2] It is meant to articulate the features shared by

[1]Illari and Williamson (2012), p. 120.

[2]The accounts synthesized by Illari and Williamson originate in Machamer et al. (2000); Glennan (2002); and Bechtel and Abrahamsen (2005).

✉ Eric Nelson Hatleback
ehatleback@cert.org; hatleback@gmail.com

Extended author information available on the last page of the article

mechanistic accounts across the sciences. Illari and Williamson note that their goal—to find the *similarities* across mechanistic accounts in the sciences—is complementary to one commonplace goal in the literature on mechanisms, which is to discern the *differences* demanded by mechanistic accounts in separate scientific fields. Accordingly, we will refer to their goal as a search for a *general mechanistic account*. Our point of departure will be the general mechanistic account generated by Illari and Williamson, labeled as follows:

> GM A mechanism for a phenomenon consists of entities and activities organized in such a way that they are responsible for the phenomenon.

The very nature of the search for a general mechanistic account makes GM continually prone to amendment. Illari and Williamson extract insight from mechanisms in astrophysics—which are notably different from the typical biological mechanisms discussed in the literature—to show how GM accommodates mechanisms across various sciences. But, as further scientific and philosophical research reveals deeper details about the mechanisms involved in still other sciences, the proposition that unifies the common features of those mechanisms will require corresponding adjustments.

The aim of this paper is to sensitize GM to features of modern computing that, to our knowledge, have not yet been addressed in the literature on mechanisms. Existing philosophical work addressing mechanisms and computing chiefly investigates whether computation is a mechanistic process. For example, Piccinini discusses whether a mechanism computes and explores what makes a mechanism a computation.[3] Whereas Piccinini's goal importantly includes application of computing to cognitive science, our emphasis rests with inspecting the merits of maintaining a strict mechanistic duality of entities and activities. Similarly, inasmuch as Piccinini is not interested in how computer scientists or programmers might benefit from mechanistic thinking, this practitioner-oriented view is a strong motivation for our work. The philosophy of technology literature does include some interaction with the biological mechanistic explanation literature, but such interaction is light. For example, early work by de Ridder does not even mention the word "computer," although it does discuss "malfunction."[4] And although Fresco and Primiero assess malfunctioning from the perspective of philosophy of computing (with a corresponding reply from Dewhurst that analyzes miscomputation), those discussions address the features of mechanisms, not the core constituents of mechanisms, and they do not engage with the wider new literature on mechanisms as we do here.[5]

A complete understanding of the constituent mechanistic parts from the perspective of computer science requires the acceptance of computer scientists *as scientists* whose goal is to better explain phenomena. As Tedre and Moisseinen make clear, even computer scientists are divided about the status of experimentation in computing.[6] In fact, many are divided about whether computer science is a science. However, Tedre's review reveals that the disagreements are more about what the interlocutors mean by "computing" and

[3] See Piccinini (2007) and Piccinini (2008).
[4] See de Ridder (2006).
[5] See Fresco and Primiero (2013) and Dewhurst (2014) for further details.
[6] See Tedre and Moisseinen (2014).

"science" than about the relationship between computer science and the other sciences.[7] Although the philosophy of technology and engineering could perhaps provide a practitioner viewpoint, modern work does not directly address computer science or mechanisms.[8] Older work, such as that presented by Simon, does in fact address computer science and mechanisms, but that work pre-dates the new mechanistic movement, so it does not offer much help presently.[9] Simon's work does, however, provide influence for the development of the new mechanistic approach: Bechtel and Richardson, for example, attribute to Simon their conception of, and strategy for, mechanism discovery as a set of heuristics.[10] The only application of mechanism discovery to experimentation in computing as positive advice for practitioners appears to be our previous work.[11] There, a division of entities and activities is classified into physical and engineered types, and the difficulty involved in designing scientifically rigorous computer science experiments based on this distinction is discussed. The present work is an orthogonal analysis that builds upon the physical/engineered mechanistic distinction for both discovery and explanation in computer science.

Our focus involves an intersection of computing and the mechanistic account that differs from the literature outlined above. Specifically, we present argumentation that demonstrates why an amendment is necessary to the ontology referred to by GM. The amendment is required due to the variability of some components in computing mechanisms: the very same component serves as either entity or activity, both between levels and within the same level of the explanatory hierarchy. We argue that the proper ontological account of these mechanistic components involves disambiguation via explicitly indexing them as entities or activities. Indexing such components yields the benefits of improved clarity of explanation and improved identification of targets for experimental intervention in domains specific to computing. We further argue, in the spirit of the general mechanistic account, that the benefits extend generally to scientific investigation beyond computing. Computing (as a whole) is not homogeneous, so the benefits of thinking about computer processes from a mechanistic perspective vary according to the purpose under consideration. For example, the software developer, the security analyst, and the program verification specialist may each perceive a different benefit from viewing computer processes mechanistically. Software developers have a well-defined concept of encapsulation as a simplifying design principle, which means, very roughly, that the developer needs guarantees on inputs and outputs of a piece of code in order to interact with it, regardless of how the code achieves the outputs. Accordingly, encapsulation shares conceptual similarities with levels of mechanistic explanation.[12] Hooking in to the mechanisms literature would give a more coherent account of the developer's explanation, understanding, and how that explanation can be refined across layers. Security analysts may gain a different benefit, such as the ability to argue (in line with Spring, Moore, and Pym) that security is a science, so long as it takes on the appropriate modern idea of scientific explanation.[13] Explaining computer

[7] See Tedre (2011).
[8] See Meijers (2009) as an example collection on philosophy of technology; computing is not addressed there.
[9] See Simon (1996).
[10] See Bechtel and Richardson (1993), Introduction (Section 2).
[11] See Hatleback and Spring (2014) for details.
[12] See Craver (2007).
[13] See Spring et al. (2017).

processes as a kind of mechanistic explanation makes linking explanations in security to explanations in other sciences much easier. The security analyst, further, could then also leverage the mechanism discovery literature. Finally, for the program verification specialist, who is concerned with ensuring the reliability of software written by others, Pym, Spring, and O'Hearn argue that the logic used to verify the software must substantially overlap with the developer's explanation of how the software works.[14] Providing links to mechanistic explanation makes the task of merging with the logic model easier by giving structure to the explanation (which otherwise seems implicit). For reliability, this ultimately means leveraging mechanism discovery to improve how flaws are found in software. Additionally, gains can be made in understanding that flaw, explaining it, and informing hypothesis generation for fixing it. Although these three example advantages of thinking about computer processes as mechanisms are not exhaustive, their diversity and importance motivates an exploration of the necessary modifications to the current mechanistic account in order to accommodate computing.

Section 2 presents argumentation, paired with an illustrative extended example from computing, for the need to revise the ontology of the general mechanistic account. In Section 3, we provide two such possible amendments and settle on indexing the components of a mechanism as the preferred solution. Section 3 also provides evidence that the proposed improvement of indexing entities and activities already has occurred implicitly in mechanistic modeling for some time, but that the habits and context of computer science make it a natural place to find the practice emphasized. Section 4 provides concluding remarks.

2 Computing and the ontological components of a mechanism

The literature on mechanisms is not unified with respect to providing an account of the ontologically distinct components of mechanisms. For example, Machamer, Darden, and Craver argue that entities and activities exhaust the ontology of mechanisms.[15] Glennan denies that activities are ontologically separate, instead maintaining that entities—and *only* entities—comprise a mechanism.[16] Illari and Williamson side with Machamer, Darden, and Craver by arguing (in part) that activities need to be recognized as ontologically separate from entities to avoid "entity bias."[17] Our intention in this section is not to argue for one ontology over another, but rather to contribute a specific example from computing that demonstrates why an amendment to GM that *indexes* entities and activities is necessary.[18]

[14] See Pym et al. (2018).
[15] See Machamer et al. (2000).
[16] See Torres (2009), where footnote 7 cites personal communication with Glennan in which he maintains that entities are the only ontological unit on mechanisms. See Glennan (1996), especially p. 53, for published insinuation that he maintains entities as the only ontologically existing components of mechanisms. Finally, see Tabery (2004) for extended discussion of the distinctive features of Glennan's position in contrast to the "dualist" position proposed by Machamer, Darden, and Craver.
[17] See Illari and Williamson (2012), pp. 126–27. See Hatleback and Spring (2014) for further discussion of the predisposition in the literature on scientific experimentation to focus on entities to the neglect of activities.
[18] While indexing the entities and activities appears to be the strongest solution, we do present an alternative solution (which involves expanding the ontology) in Section 3.

Elsewhere, we have argued that scientific experimentation in computing breaks into two distinct classes: one class focuses on physical mechanisms and the other class focuses on engineered mechanisms.[19] Computing is a discipline largely focused on creation and engineering. Engineered mechanisms, by contrast with physical mechanisms, are changeable in practice at the will of some decision maker. Practical challenges in sub-disciplines like computer security are best understood, in our estimation, by marking a heuristic distinction between physical mechanisms and engineered mechanisms. Furthermore, the properties of engineered mechanisms are not unique to computing, and they bring to the fore an aspect of mechanisms that GM (in its present form) does not accommodate.

Engineered mechanisms in computing include manipulable components such as software and computer code. We argue that these mechanistic components are neither strictly entities nor strictly activities. We will refer to such components of mechanisms as *variable* or *variable components*.[20] In order for GM to accommodate variable components, such as the engineered mechanisms that emerge in computing, an update is required.[21] In tandem with our argumentation, we will develop—in greater and greater detail as the argumentation proceeds—a running example involving a file on a computer to illustrate the necessity of amending GM in accordance with mechanisms in computing.

Consider, first, a basic file. Perhaps it is the file that contains the digital version of this essay. At one organizational level, the file is an entity: a file is a thing, and activities happen to that thing. It can be deleted, for example. However, the "delete" activity is accomplished via another file, located elsewhere on the computer, which stores the instructions for the "delete" activity. Software is stored in files, and data is stored in files.

Concerning the "delete" file, then, it is contextual whether the file is an entity or whether it is an activity. The file's ontological status appears to be unclear. Neither its status as software, nor its status as data, takes obvious precedence. A context in which to consider the file is required. We must avoid the usual prejudice that nouns are entities

[19] See Hatleback and Spring (2014).

[20] In Section 3, we distinguish two possible alternative accounts of the variability: one treats the variability as a feature of entities and activities, and the other treats variable components as ontologically distinct from entities and activities. The former emerges, in part, from the literature on the explanatory virtues of the mechanistic account. For example, Glennan's Law, which maintains that a mechanism is at least partly individuated by the phenomenon in question, appears to suggest that variability in the entities and activities is merely a feature of those entities and activities. And, as Dupré notes (Dupré (2012), p. 30), "attempts to understand phenomena at a particular organizational level determine schemes of classification at that level." Dupré's approach suggests that the theorist, by choosing the level of explanation at which to investigate, also implicitly chooses the classification of the mechanism's components at that level. In what follows, we will suggest that an explicit classification scheme for variable components further strengthens the mechanistic account.

[21] It is possible—although fuller argumentation would be required to establish the possibility—that the update to GM to accommodate variable components might simultaneously resolve a tangential issue. The ontology of static entities and activities in the current formulation of GM leaves open disputes between those who favor models as composed predominantly of processes or activities and those who favor models as composed predominantly of entities and their properties. See Dupré (2012), p. 30, for an example of the former position. See Glennan (2005), p. 445, for example of the latter position, where he notes that "mechanisms underlie behaviors," but the behavior is the "'phenomenon' that the mechanism produces," rather than an activity that is a component of the mechanism. Permitting the indexing of variable components (as will be urged in what follows) could resolve this complication by enabling each side of the dispute to build into the indexing the qualities of the other side's position.

and verbs are activities; "a file" may be suggestive that the "file" is a noun, but the metaphor describing the computer code as "a file" is potentially blinding. It is not an entity in the same unambiguous way that a piece of paper in a filing cabinet is a file, which is what the metaphor falsely suggests. The symbolic instructions do not change when the stored "delete" file is loaded and executed to delete the file containing our manuscript. The set of symbolic instructions appears to be an entity when stored on disk as a file, but it also appears to be an activity when it is read by the processing chip. There is an important difference between the "delete" program's file on disk, which cannot destroy files, and the activity of deletion, which does destroy files. Yet both the entity and the activity refer to the very same series of instructions. In terms of its role in a mechanism, this component is variable; its status as entity or activity varies according to the context in which it is considered.

2.1 An extended example: Further details of computing mechanisms

To demonstrate the issue, consider still finer details of the example begun in the previous paragraph. At the mechanical level, data is stored on a computer disk as a pattern of electromagnetic signals. These signals represent bits, which are a physical representation of a mathematical notion of information which has been implemented, essentially unchanged, since the late 1940s.[22] A computer processes these electromagnetic signals in its central processing unit (CPU), using complex combinations of relatively simple instructions encoded by the execution of individual electronics inside the CPU. Example instructions might include: read the value of the number in *this* location, write a number to *that* location, or perform addition on *these* two numbers. All of these instructions, and thus modern computers themselves, are physical realizations of the mathematical model developed by Turing in the 1930s.[23]

Given these details, one might surmise that an electromagnetic pattern is an entity when stored on a computer disk and an activity when used in a CPU register, which thereby appears to surmount the obstacle. But this approach unfairly glosses over serious problems caused by the indistinguishability between an entity and an activity in computer science. The problem emerges, for example, in computing program verification tasks involved with security and reliability. There, it is referred to as "co-mingling data and control instructions"[24] or "improper input validation."[25]

Computer security involves the abstract interaction of the defender and the adversary. To exploit "improper input validation" successfully, the adversary's general strategy is to cause data (an entity) of the adversary's choosing to be executed as code (an activity) in order to take control of the computer. The defender's goal, conversely, is to distinguish accurately input data from the defender's own code. This small battle is fought, with the website as the defender, every time someone enters a username and

[22] See Shannon and Weaver (1949).
[23] Turing (1936).
[24] IEEE (2015). Note that IEEE, as the relevant trade association and international standards body, is authoritative for definitions of problems for practitioners in this space.
[25] MITRE (2008). MITRE maintains the CWE and CVE definitions and repositories under contract to the US federal government, and within the scope of US government operations and policy, those definitions are authoritative. MITRE's CWE and CVE work is also the de facto global standard for definitions of software vulnerabilities.

password or posts content on a social media website. The input validation problem persists, and attacks continue to be successful. Adversaries succeed, despite the fact that Microsoft, Apple, Sun, and the open-source community all implement sophisticated defenses in modern operating systems to reduce the security impact when programs fail to identify which entities should not be executed as activities.[26] The multiplicity of levels of abstraction routinely used in computing exacerbates the problem of separating code (activities) and data (entities).[27] An essential functionality of a modern computer is the flexibility to treat data as code and code as data. Because of this, the problem of indexing entities and activities is not an abstract or rare occurrence in computing. Computer scientists, engineers, and software developers are trained habitually to index whether something is an entity or an activity—whether it is data or code.[28]

Proceeding further, consider the common human-user-level instruction "delete essay.txt" for removing the file containing this essay from the computer's hard drive. While processing the command, various components in the mechanism associated with the phenomenon oscillate from entity to activity multiple times in the milliseconds before the instruction is executed. First, the whole phrase is received by the computer interface as data, an entity. The example assumes that the data is typed in symbols, such as `shred essay.txt`, although instead it could be delivered in other forms, for example by clicking on the file's icon and pressing the delete key on the keyboard. Second, by convention, the entity "shred essay.txt" is split on the first space character into the command (activity) and the parameters for that command (entities). However, computer commands need not have only one such splitting between entity and activity. It is quite natural to write `sudo shred essay.txt`, in which case the entity is split once, into the "sudo" command and the parameters "shred essay.txt". The effect of the "sudo" activity is to execute its parameters as a command, but as a different user with a higher level of access. Thus, the entity "shred essay.txt" immediately becomes an activity, "shred", and an entity, "essay.txt". There are

[26] See the statement made by the United States National Security Agency (NSA) at https://www.nsa.gov/ia/_files/factsheets/i733-tr-043r-2007.pdf. At the time of the statement, NSA's responsibilities included defense of all US government military computer systems. Their statement that all operating systems at the time suffered from this failure to differentiate code and data (indexing activities and entities in our language) should be considered a definitive statement on the gravity of the security impact of this failure. The defenses are, namely, Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR).

[27] Modern computing is only plausible, in terms of the day-to-day engineering and maintenance of actual systems, because each component of a computer provides an abstract method by which other components interact with it. This "modular design" permits human designers to learn how to use a component effectively without needing to understand exactly what it does and how. A standard laptop is exemplary concerning the levels of abstraction involved in the computing process. The laptop exhibits a procession of perhaps 10 levels of abstraction, each of which interacts almost exclusively with the level immediately before it and after it. Step by step, the interactions build from the electricity flowing through transistors on the physical chip up to the human user's interaction with the observable results. Each of these interactions is potentially a many-to-many relationship. For example, a human user could use different applications simultaneously; or the power supply component could distribute electricity to the processor, the USB (universal serial bus) ports, the graphical processing unit (GPU), and the data storage disk (hard drive) simultaneously. At each of these interfaces, what counts as code or data often changes, and the change is by design. The flexibility is what makes modern computing possible. But the flexibility also causes the difficulty of accurately distinguishing code and data. Because of this, the problem of indexing entities and activities is not a rare occurrence when working with computers.

[28] This habit is aspirational, of course, and many people in the field do not implement the distinction perfectly. However, see the earlier-referenced sources for examples where the authoritative organizations state the aspiration.

physical bounds, based on the computer's storage capacity, on the number of times an input command could be split into entity and activity while maintaining that the data (entity) be considered a new command consisting of its own entity and activity.[29] However, from the perspective of the theory and design of programming languages, commands are defined inductively and parsed recursively, so most modern computer commands are defined such that they can (in principle) be split in this manner infinitely many times.[30] Therefore, one cannot *prima facie* determine which segments of a command are entities and which are activities; the ambiguity, or "flexibility," as a programmer may prefer to call it, is built in to computer systems.

The only physical extension with which to identify these commands and their constituents is the pattern of bits electromagnetically stored by the computer. While it is true that this essay has an orthographic form that you, the human, are reading presently, there is no commonly understood meaning for a computer's operation that references anything other than a stored electromagnetic pattern. All computers are physical instantiations of the mathematical model that Turing proposed to determine the computability of certain rational numbers, the eponymous Turing Machine.[31] Turing describes the machine's internal configuration and how that interacts with a "tape"—a stream of symbols the machine reads and writes based on the machine's internal configuration. Although this suggests that activities might be identified with the Turing Machine's internal configuration and entities might be identified with the symbols on the tape, Turing demonstrates that any Turing Machine can be simulated by just one machine where the first instructions on the tape are constructed such that they represent the internal configuration of the machine to be simulated. Accordingly, the symbols on the tape cannot be identified *merely* as entities, since the symbols are capable of encompassing the full scope—including the activities—of an entirely separate Turing Machine. Common parlance for such a "multi-level" Turing Machine is "Universal Turing Machine" (UTM). The hardware physically fixed in silicon in a contemporary computer is a UTM. The UTM does not know *prima facie* whether any stored symbols—that is, any given electromagnetic pattern—represent i) instructions for configuring its state, or ii) data to process according to that state. Note that even this process of simulating a UTM is recursive; the instructions to reconfigure the internal state are not privileged. The recursiveness of the underlying theory of computation mirrors the infinite recursiveness in programming language design. Thus, even at the most fundamental level, the distinction between entities and activities in computing is blurred. The flexibility to treat data as code is what makes a UTM universal rather than specific.

2.2 Two potential objections

We turn now, briefly, to addressing two potential objections to the claim that variable components are distinctly unique from their non-variable counterparts. First, it might be objected that the files "shred" and "essay.txt" (for example) are clearly distinct: they are objectively different patterns of electromagnetic binary digits (bits). That is, perhaps the

[29] For example, a typical laptop at the time of publication has about 4–8 GB of RAM and 0.5–1 TB of hard drive space. Thus, the number of commands a commodity laptop could hold and parse is on the order of hundreds of millions to hundreds of billions. The practical limitations, therefore, are irrelevant for the present discussion.

[30] See Winskel (1993).

[31] See Turing (1936).

properties "entity" or "activity" are embedded somehow in the patterns, and such embedding has yet to be articulated in the literature. Second, it might be objected that it is a trivial task to split a purported variable component into subsets, as described in the "delete essay.txt" portion of the example provided earlier, such that one subset clearly is an entity and the other subset clearly is an activity. Although Turing's result (namely, that the parsing of the command is not generally decidable) is quite strong, it may be the case that the concept of termination or parsing in computing does not match with the concept of partitioning a model into entities and activities within a particular level of mechanistic granularity.[32] To reply to these objections, we return to the development of the example that has accompanied the argument thus far to demonstrate that it is not the case that the properties "entity" and "activity" are embedded in the patterns of bits representing the files, nor is it the case that intuitive differentiation within one level of analysis is tenable.

Consider typing `/bin/shred /bin/shred` as a command.[33] The location "/bin" is where a Linux computer stores the executable programs. The object "/bin/shred" uniquely identifies exactly one file on the computer: the file that contains the instructions for shredding files. The computer splits the human-input entity "/bin/shred /bin/shred" into two parts, the activity "/bin/shred" and the entity "/bin/shred". This is not an instance where the computer changes the program when moving it from hard drive storage to memory for execution.[34] The source code is first interpreted by a compiler and translated into a set of machine-readable instructions that are particular to the architecture of transistors and other silicon features of the computer chip.[35] It is compiled into the binary instruction set.[36] Accordingly, the set of files under discussion

[32] Turing's result has been proven equivalent in relevant ways to approaches by Gödel (see Gödel (1931)) and Church (see Church (1936)). Additionally, one might advance this objection despite the fact that Bechtel and Richardson explicitly develop their mechanistic approach to scientific problem solving as inspired by problem space search heuristics in computer science stemming, ultimately, from Turing's results. See, for example, Bechtel and Richardson (1993, p. 12), where they claim: "Analogously [to problem solving as defined in computer science and artificial intelligence], the task of constructing an explanation for a phenomenon in a given scientific domain is one of finding a sufficient number of variables, the constraints on the values of those variables, and the dynamic laws that are functions on those variables, so that it is possible to predict future states of affairs from descriptions of the universe at an earlier time." The link to Turing's work is implicit in the connections to computer science, which surface in the relevant artificial intelligence literature as "bounded rationality," as coined by Herbert Simon.

[33] Shred represents the secure part of the secure delete process. Shred overwrites the contents of a file with random content to destroy it. The analogy with a physical paper shredder is that both make documents in the trash unrecoverable.

[34] There are instances in computing where this would appear to happen, for instance compiling source code into binary code. Source code is the human-readable commands programmers use to construct and design programs. For example, the last line of the source code of the GNU version 8.24 (i.e. the default Linux version) of the shred program is `` `return ok? EXIT_SUCCESS: EXIT_FAILURE;' ``. One does not need to understand exactly what it means to gather the intended intuition that source code is a language accessible to humans. Computer processors do not directly interpret these lines.

[35] Modern computer chip architectures are largely standardized to a few options, which makes it possible for users to avoid this step completely. Although the step is invisible to most users, it still always happens at some point.

[36] Presenting a binary instruction set requires special notation. There are not "lines" of binary, so there is no analogous last line of the compiled code. However, for the sake of comparison, if electromagnetic polarities are represented by bits 1 and 0, then the last 40 bytes of the example given in footnote 34 (we presented 40 characters of source code, and one character happens to be one byte) of the shred version 8.24 binary code are a rather boring 320 zeros (one byte has eight bits).

in "/bin" are just those which already have been transformed into exactly the form that the processor executes when it does so. There may be interesting questions about the relationship between a binary instruction set and the source code from which it derives, but our example focuses only on the compiled binary code.

It is clear, then, that the two instances of /bin/shred, the command and the argument, are not merely homographs. In the command `/bin/shred /bin/shred`, the first instance does not stand for a subtly different part of electromagnetic storage from the second, nor does one refer to a set of symbols that is different from the other, as would be the case if one were source code and the other were binary instructions. They are not the same orthography referencing different physical extensions in the world. Instead, both the entity and the activity are an identical pattern of electromagnetic storage that is co-extensive in space at the time of typing the command.

Despite the oddness of this duality, a standard Linux computer will successfully complete the command `/bin/shred /bin/shred` without any difficulty or special setup. The user merely needs to satisfy the usual requirement of having permission to change the file. The reason that this works is due to the designed relationship between a CPU and the storage disk. The CPU can read a copy of the binary instructions from the file in storage at "/bin/shred" and store a copy of them in CPU memory. This duplication of the same set of instructions allows the computer to execute those very instructions to shred the file "/bin/shred" from which it read the instructions. One object, in one everyday phenomenon, has a variable role in this case. There is no underlying encoding for "entity" or "activity" present in the pattern of bits, nor can one intuitively differentiate the "entity" portion of the pattern from the "activity" portion in instances where the pattern instantiates a variable mechanistic component.

3 Amending the general mechanistic account

We turn now to providing what we take to be the most plausible philosophical stance that one might adopt with respect to updating the general mechanistic account in accordance with the evidence provided by the variable components of engineered mechanisms in computing. The update involves modifying GM so that the variable components are explicitly indexed as entities or activities. The ontology itself is left unaffected, since the variable components still remain either entities or activities. However, the *implementation* of the ontology is affected, since certain components of mechanisms (the variable components) require attention in that they be labeled—*indexed*—as entities or activities, which is a step not commonly applied to the "standard" components of mechanisms (i.e., the non-variable entities and activities). The amended GM reads:

> GM (indexed): A mechanism for a phenomenon consists of *indexed* entities and activities organized in such a way that they are responsible for the phenomenon.

It may seem that the distinction exemplified by the file deletion case warrants nothing more than careful consideration when dealing with computing mechanisms. For example, it may seem that the context or location of the instruction set in those instances clearly determines whether the component in question is an entity or activity: if a file is

on the hard drive, it is an entity, and if it is in the CPU chip, it is an activity. However, note that this objection itself establishes the legitimacy of the proposed amendment to GM; it is *because* the ontological status of a variable component is contextual that GM requires an amendment.[37] If the status of the binary pattern stored in the file /bin/shred (for example) were *de re* something particular—entity or activity—regardless of context, then GM (indexed) would be unnecessary. Indeed, in that case, no mechanistic components would be variable. In light of the argumentation supplied in Section 2, we do not believe this to be the case.

In practice, the notation for indexing might involve, for example, attaching the status of variable components (such as code and software) to each reference of the variable component on a diagram of the mechanism's operation. The index attached to variable components (that is, whether they act as entities or as activities) often is dependent upon the level of the mechanism under investigation within the explanatory hierarchy. When the phenomenon in question arises from a mechanism that takes code (for example) as an entity, then the software would be identified as "software$_e$". Likewise, when the phenomenon in question arises from a mechanism that takes the software to be an activity, it would be identified as "software$_a$". The notation suggested for implementing GM (indexed) is consistent with, but more precise than, the notation that would be employed when implementing GM, which involves the identification of entities and activities in an absolute fashion. When implementing GM (indexed), the selection of entities and activities occurs with respect to the level within the explanatory hierarchy of the phenomenon in question. For example, if the user deletes the file that contains the "delete" instructions, the mechanism could be modeled as depicted in Fig. 1. The indexing of the variable component "delete" is explicit in Fig. 1. The same name is maintained for the variable component to indicate that it is the same in important respects. In this particular case, "delete$_a$" and "delete$_e$" are the same in that they are represented by exactly the same set of symbols. The argumentation provided in Section 2 demonstrates how the very same item—such as the computer file "delete"—can separately play the role of both entity and activity.

One might wonder whether there exists a 'correct' indexing for the components of a mechanism. Alas, there is no one 'correct' indexing; the indexing is contextual. It may be possible to link this context to the level of abstraction described by Floridi.[38] But it lies beyond our scope to try to link the mechanistic literature to formal systems such as logic (in this case, Floridi's logic of information).[39] Because computer science is a diverse field with diverse concerns, it is unlikely that one answer will suffice for generating the best level of abstraction from which to view mechanism components. Networking specialists, for example, might view whole computer hosts as entities, whereas operating systems specialists seek to explain the behavior of the host as the phenomenon built up of software and hardware. Extensive future work lies ahead in

[37] The difficulty (or ease) of indexing components clearly lies beyond the scope of the present work. Rather, we wish to note merely that the apparent objection is an intuitive and direct way to establish the need for the indexing we propose.

[38] See Floridi (2011) for details.

[39] Pym et al. (2018) describe a case where scientific and logic models are merged in computer science, specifically program verification. So it does appear to be possible to link a mechanistic and logical account of a phenomenon in computer science, but it is painstaking work to merge them.

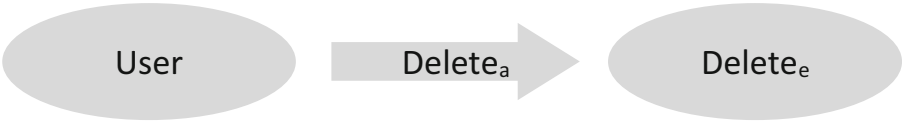

Fig. 1 A simple mechanism with variable entities and activities indexed

terms of describing how level-of-abstraction choices influence mechanism discovery and explanation in each of these sub-fields of computing.

3.1 Considering alternative refinements

Future research into mechanisms from other scientific disciplines likely will contribute feedback that will indicate whether indexing the variable components is the proper refinement to GM. Other options remain available. For example, the general mechanistic account could adopt an expanded ontology that includes variable components as altogether separate ontological units. In such a case, GM would be supplanted not by GM (indexed), but instead by something like:

> GM (expanded): A mechanism for a phenomenon consists of entities, activities, and variable components organized in such a way that they are responsible for the phenomenon.

The added piece of ontology in GM (expanded) indicates that variable components such as software occupy an ontologically separate category from entities and activities. Expanding the ontology in this fashion would impact directly the study of mechanisms in the philosophy of science literature, since the properties of the new ontological unit would invite extended examination. Additionally, the expanded ontology would have direct implications for scientists and experimenters, since they would need to account for these new variable components as a separate part of experimental design.

If treated this way, executable files on a computer, such as the "delete" file, would be simply considered variable components—neither entities, nor activities. Recognizing variable components as ontologically separate from entities and activities might, in fact, provide a partial explanation for the difficulty faced in the computer science literature attempting to adopt the methods of other sciences: computer science deals extensively with executable files, so it is disproportionately affected by variable components, which have not received detailed study in the mature sciences.[40] Nonetheless, granting separate ontological status to variable components presently seems too grand, since it is not (yet) clear what is gained by such an approach beyond classificatory simplicity. Instead, GM (indexed) supplies the amendment necessary to update GM with the ability to accommodate mechanisms involving variable components.

Further, and of practical importance, the switch can be informative for experimental design in sciences where variable components are commonplace in the mechanisms under study.[41]

[40] See, for example, Schiaffonati and Verdicchio (2014) for an account of the difficulty faced by computer science in adopting the methods of other sciences.

[41] See the brief comments at the close of this section for suggestions regarding the potentially positive impact for experimental design that is afforded by the switch to GM (indexed).

3.2 Indexing beyond the science of computing

It is worth pursuing one further potential objection to the refinement of GM to GM (indexed). One might argue that GM (indexed) is merely a special case of GM that is relevant only to computing, rather than a generalization that is relevant to science more broadly. The objection might continue that only variable components from computing need to be indexed, that other sciences do not have such components, and (therefore) that the bulk of scientific investigation and explanation is captured accurately with GM.

The relevant reply to this objection is twofold. First, in a general sense, the inclusion of (perhaps) idiosyncratic qualities from diverse sciences supervenes on the very search for a general mechanistic account. That is, the endeavor itself is an inclusionary exercise that proceeds only via careful consideration of unique facets emerging in individual sciences.[42] Second, and particular to the case at hand, the benefit of adopting GM (indexed) involves the ability to extend the indexing concept to variable components that might arise in the future in other sciences (or to existing mechanistic pieces that have not heretofore been identified as variable components).[43] This flexibility comes at zero cost to familiar mechanistic accounts that do not involve variable components. Such accounts remain unchanged, since they contain no variable components to index.

Indeed, one advantage of updating the general mechanistic account by indexing the variable components is that, in some cases, the status of a mechanism's components can be an indicator that guides a scientist's experimental design. For a particular example, consider computer security, which is the subdiscipline of computer science concerned with simultaneously ensuring the confidentiality, integrity, and availability of information technology systems against the attacks of adversaries.[44] Practitioners find value in knowing whether an element of an attack is an entity or an activity; the knowledge helps in the process of devising the proper defense to be deployed.[45] In our experience, defenses are readily interpreted as entities, while attacks are activities, in the mechanism that is responsible for the adversary-defender interaction phenomenon. The underlying cause of this division is an area of future research. However, regardless of

[42] The clearest example of this is GM itself, which Illari and Williamson construct after careful consideration of mechanisms arising in astrophysics.

[43] Two examples include cellular biology and chemistry. Each requires a deeper account than space permits here, but the following brief digression provides a rough sketch of the role GM (indexed) might play in those sciences. In the setting of cellular biology, when a protease enzyme catalyzes a proteolysis reaction to break down the chemical bonds of another protease enzyme of the same type, the enzyme is both the source and the target of the activity in an important sense. In the context of chemistry, the mechanism responsible for the phenomenon of chemical bonding involves the transfer of electrons between atoms. Two atoms bond when electrons—particles, entities—are transferred from one atom to the other. The bond between the atoms occurs via the interaction between their electron shells. Electron shells, however, are mechanistically depicted as diffuse clouds or fields of force, not particulate electrons. Chemists regularly discuss the resulting bond as an object, but another accurate characterization of the chemical bond is the merged field of force of electron shells maintained by the interaction of electrons. Although it is not common parlance, the bond might be viewed instead as *electrons as activities*, as it were, that give rise to the bond. These two examples, although trivial, suggest that the explicit indexing of variable mechanistic components in sciences beyond computing could be explanatorily useful, but it has not yet proven necessary, because scientists already perform some semblance of indexing naturally and intuitively.

[44] For a high-level, strategic introduction to computer security see Shimeall and Spring (2013).

[45] See Spring and Hatleback (2017) for a detailed account.

the underlying cause, the heuristic is useful for designing effective experimentation. Experimentation aimed at demonstrating the existence of an entity is designed differently from experimentation aimed at demonstrating the existence of an activity. Accordingly, in the case of experimentation aimed at understanding variable components, an initial step in the experimental design process would involve identifying the proper index for the variable component. The literature in the field reveals the potential impact of this change, since many research papers focus on demonstrating the existence of a particular attack or class of attacks.[46] Proper indexing of the mechanistic components could prove immensely helpful in those instances.

4 Conclusion

The result of the preceding is an amendment to the general mechanistic account to account for mechanisms that are prevalent in computing. GM, as put forth by Illari and Williamson, has been updated to GM (indexed). The complete, updated general mechanistic account thus reads:

> GM (indexed) A mechanism for a phenomenon consists of indexed entities and activities organized in such a way that they are responsible for the phenomenon.

This result is a general mechanistic account that accommodates both the nuances of the variable components included in engineered mechanisms in computing and the properties of astrophysical mechanisms investigated by Illari and Williamson in their original proposal of GM.

Because variable components are less prevalent than "standard" entities and activities, the indexing process does not introduce a drastic change to the task of mapping mechanisms. Nonetheless, indexing is necessary because the maps of mechanisms might include a variable component that is listed as an entity at one step of an explanatory hierarchy while simultaneously appearing as an activity at a different step. Figure 1 displays a simple example of this notational clarity: there, the "delete" file appears as both entity and activity within the same phenomenon. Across different phenomena or across levels of explanation, clear notation should help prevent confusion in professional communication. Indexing also potentially expands the possibilities for targets for intervention in a mechanism. Finally, clarity can have direct practical implications, as well, for scientists in their experimental design.

It is to be expected that further amendments to GM will occur as the mechanistic accounts implemented by additional sciences are taken into consideration. Our contribution has been to update the general mechanistic account provided by Illari and Williamson so that it properly reflects engineered mechanisms of the sort that arise in computing.

[46] For example, 33 of the 67 papers in one of the premier information security venues—the 24th USENIX Security Symposium, August 12–14, 2015— have such a focus. (Those papers with "proof of existence" emphasis begin on pages 17, 81, 97, 113, 129, 161, 177, 193, 255, 271, 287, 399, 463, 563, 579, 595, 611, 627, 643, 659, 707, 723, 769, 785, 801, 833, 849, 897, 925, 945, 977, 1009, and 1025).

Acknowledgements The authors are grateful to Phyllis Illari for discussion and insightful comments on early drafts of this work.

References

Bechtel, W., & Abrahamsen, A. (2005). Explanation: a mechanist alternative. *Studies in History and Philosophy of Biological and Biomedical Sciences, 36*, 421–441.

Bechtel, W., & Richardson, R. C. (1993). *Discovering complexity: Decomposition and localization as strategies in scientific research*. Princeton: Princeton University Press.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics, 58*, 345–363.

Craver, C. (2007). *Explaining the brain: Mechanisms and the mosaic of unity in neuroscience*. Oxford: Oxford University Press.

de Ridder, J. (2006). Mechanistic artefact explanation. *Studies in History and Philosophy of Science, 37*, 81–96.

Dewhurst, J. (2014). Mechanistic miscomputation: a reply to Fresco and Primiero. *Philosophy & Technology, 27*, 495–498.

Dupré, J. (2012). *Processes of life: Essays in the philosophy of biology*. Oxford: Oxford University Press.

Floridi, L. (2011). *The philosophy of information*. Oxford: Oxford University Press.

Fresco, N., & Primiero, G. (2013). Miscomputation. *Philosophy & Technology, 26*, 253–272.

Glennan, S. S. (1996). Mechanisms and the nature of causation. *Erkenntnis, 44*, 49–71.

Glennan, S. (2002). Rethinking mechanistic explanation. *Philosophy of Science, 69*, S342–S353.

Glennan, S. (2005). Modeling mechanisms. *Studies in History and Philosophy of Biological and Biomedical Sciences, 36*, 443–464.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für mathematik und physik, 38*, 173–198.
Hatleback, E., & Spring, J. M. (2014). Exploring a mechanistic approach to experimentation in computing. *Philosophy & Technology, 27*, 441–459.
IEEE. (2015). Strictly separate data and control instructions, and never process control instructions received from untrusted sources. Institute of Electrical and Electronics Engineers. http://cybersecurity.ieee.org/2015/11/13/strictly-separatedata-and-control-instructions-and-never-process-control-instructions-received-from-untrusted-sources/. Accessed 15 August 2016.
Illari, P. M. K., & Williamson, J. (2012). What is a mechanism? Thinking about mechanisms *Across* the sciences. *European Journal for Philosophy of Science, 2*, 119–135.
Machamer, P., Darden, L., & Craver, C. (2000). Thinking about mechanisms. *Philosophy of Science, 67*, 1–25.
Meijers, A. (Ed.). (2009). *Philosophy of technology and engineering sciences*. Amsterdam: North-Holland.
MITRE Corporation. (2008). CWE-20: Improper Input Validation. Common Weakness Enumeration, definitions version 2.9. https://cwe.mitre.org/data/definitions/20.html. Accessed 15 August 2016.
Piccinini, G. (2007). Computing mechanisms. *Philosophy of Science, 74*, 501–526.
Piccinini, G. (2008). Computation without representation. *Philosophical Studies, 137*, 205–241.
Pym, D., Spring, J. M., & O'Hearn, P. (2018). Why separation logic works. *Philosophy & Technology*. https://doi.org/10.1007/s13347-018-0312-8.
Schiaffonati, V., & Verdicchio, M. (2014). Computing and experiments: a methodological view on the debate on the scientific nature of computing. *Philosophy & Technology, 27*, 359–376.
Shannon, C. E., & Weaver, W. (1949). *The mathematical theory of communication*. Urbana: University of Illinois Press.
Shimeall, T., & Spring, J. M. (2013). *Introduction to information security: a strategic-based approach*. Waltham: Elsevier.
Simon, H. A. (1996). *The sciences of the artificial* (3rd ed.). Cambridge: MIT Press.
Spring, J. M., & Hatleback, E. (2017). Thinking about intrusion kill chains as mechanisms. *Journal of Cybersecurity*, tyw012. https://doi.org/10.1093/cybsec/tyw012.
Spring, J., Moore, T., & Pym, D. (2017). Practicing a science of security: a philosophy of science perspective. *Proceedings of the New Security Paradigms Workshop*, October 1–4, Santa Cruz, CA, Unites States of America.
Tabery, J. G. (2004). Synthesizing activities and interactions in the concept of a mechanism. *Philosophy of Science, 71*, 1–15.
Tedre, M. (2011). Computing as a science: a survey of competing viewpoints. *Minds and Machines, 21*(3), 361–387.
Tedre, M., & Moisseinen, N. (2014). Experiments in computing: a survey. *The Scientific World Journal, 2014*, 1–11. https://doi.org/10.1155/2014/549398.
Torres, P. J. (2009). A modified conception of mechanisms. *Erkenntnis, 71*, 233–251.
Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 42*, 230–265.
Winskel, G. (1993). *The formal semantics of programming languages: An introduction*. Cambridge: MIT Press.

Affiliations

Eric Nelson Hatleback[1] · **Jonathan M. Spring**[1,2]

[1] Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

[2] University College London, London WC1E 6BT, UK