Ergalics: A Natural Science of Computation

Richard T. Snodgrass
Department of Computer Science
University of Arizona
Tucson, AZ
rts@cs.arizona.edu

February 12, 2010

Abstract

If computer science is to truly become a "science," as Newell, Perlis, and Simon originally conceived, then it must integrate three equally ascendant perspectives: mathematics, science, and engineering. The scientific perspective can help us to *understand computational tools* and *computation* itself, through the articulation of *enduring*, *technology-independent* principles that uncover new approaches and identify technology-independent limitations. This reorientation will yield an understanding of the tool *developers* and the tool *users* and will thus enable refinements and new tools that are more closely aligned with the innate abilities and limitations of those developers and users. Incorporating the scientific perspective, to augment the mathematical and engineering perspectives, necessitates that the discipline study different phenomena, seek a different understanding of computation, ask different questions, use different evaluative strategies, and interact in different ways with other disciplines. Doing so will add wonder, engagement, and excitement to our discipline.

1 Introduction

Three quite distinct perspectives—mathematics, science, and engineering—have long been associated with the discipline of computer science [Denning 2005a]. Mathematics appears in computer science (CS) through formalism, theories, and algorithms, which are ultimately mathematical objects that can be then expressed as computer programs and conversely. Engineering appears through our concern with making things better, faster, smaller, and cheaper. Science may be defined as developing general, predictive theories that describe and explain observed phenomena and evaluating these theories [Aicken 1984, Chalmers 1999]. Such theories include a statement of "why" and are amenable to predictions on heretofore unexamined phenomena, that can be subsequently tested on those phenomena [Davies 1973].

In an influential letter in the journal *Science*, Nobel Prize winner Herbert Simon and his colleagues Alan Newell and Alan Perlis defined the nascent discipline of CS as a "science of the artificial," strongly asserting that "the computer is not just an instrument but a phenomenon as well, requiring description and explanation" [Newell, Perlis, and Simon 1967, page 1374]. Simon furthered that line of reasoning in his book, **Sciences of the Artificial**, whose third edition was published in 1996.

Simon's promise of a science of computational phenomena has not yet been realized by this fortieth anniversary of his compelling argument. Since its founding in the late 1950's, CS has uncovered deep mathematical truths and has constructed beautifully engineered systems that have ushered in the "age of information technology." However, there is as yet little understanding of these highly complex systems, what commonalities they share, how they can be used to construct new systems, why they interact in the ways they do with their environment, or what inherent limitations constrain them. As an example, while solid-state physics has provided a detailed understanding of VLSI, enabling design rules that allow the construction

of highly complex processors, there is no similar understanding of the efficacy of programming language constructs and paradigms that would allow programming language designers to predict the efficiency of code written in their new languages. Nor do we have a detailed understanding of the failure modes of software components and systems assembled from those components, an understanding that would allow us to predict failure rates and to engineer systems to predictably decrease those rates.

The attainments of CS are largely the result of three contributing factors: the exponential growth of computing, storage, and communication capacity through Moore's Law; the brilliance of insightful designers; and the legions of programmers and testers that make these complex systems work by dint of sheer effort. Unfortunately, CS is now constrained in all three factors: the complexity of individual processors has leveled off, the systems are at the very edge of comprehension by even the most skilled architects, and the programming teams required for large systems now number in the thousands. Development "by hook or by crook" is simply not scalable, nor does it result in optimally effective systems. Specifically, unlike other engineering disciplines that have solid scientific foundations, CS currently has no way of knowing when a needed improvement can be attained through just pushing a technology harder, or whether a new technology is needed. Through a deeper understanding of computational tools and computation itself, ouir discipline can produce new tools that are more robust, more effective, and more closely aligned with the innate abilities and limitations of the developers and users.

CS has not yet achieved the structure and understanding that other sciences aspire to and to which the public ascribes the ascendancy of science. Consider as just one example how drugs now can be designed based on their desired shape, which was enabled by deep understanding of protein folding, chemical bonding, and biological pathways. This understanding is due primarily to the adoption of scientific reasoning. To achieve such understanding of computational tools and computation, the mathematical and engineering perspectives, while critical, are not sufficient. Rather, what is needed is the application of a radically different perspective, that of the scientific method, to enable and test scientific theories of computational artifacts. (We will shortly examine one such scientific theory, Denning's Theory of Locality.) Adopting such a perspective within CS, to augment the existing, vital perspectives of mathematics and engineering, requires that we study different phenomena, seek a different understanding of computation, ask different questions, use different evaluative strategies, and interact in different ways with other disciplines.

2 Studying Different Phenomena

CS has been prolific and influential in creating computational tools that allow human intelligence to be amplified. Computational tools, defined broadly, include formalisms (e.g, order notation), conceptual devices (e.g., UML), algorithms, languages (e.g., Java, SQL), software systems, hardware (e.g., RISC), and computational infrastructure (e.g., the internet).

The central premise of this essay is the following.

> *The focus of the discipline of CS over its first fifty years has been to build larger and larger assemblages of algorithms, data structures, and individual programs into massive, complex, and highly-useful systems. These computational artifacts are inherently deserving of study. These tools are sufficiently stable that meaningful statements about them are possible. Studying the tools themselves, their behavior, their structure, and how users interact with them, as well as studying computation itself, will yield insights that will enable nascent scientific theories that can serve to improve the tools in fundamental ways, to explain fundamental limitations, and to understand the role of computation in nature.*

This is admittedly an audacious claim. Is it just wishful thinking? The purpose of this essay is to characterize a science of computation and to explain how it can transform CS. Our argument has the following form. Science provides a different understanding than mathematics or engineering: that of general, testable, predictable scientific theories. It is these theories that have enabled the great scientific advances of the last century. There are several fundamental reasons why theories of computation can be as enduring and general as the theories concerning other aspects of nature as articulated by other sciences. A science of computational tools and of computation generally could achieve advances of similar import by utilizing the methodology that other sciences use to generate and test their theories, specifically, that of empirical generalization and model testing. Both the history of natural science and the history of CS provide strong predictors of disruptive innovations that could be enabled by a vigorous discipline of science within CS.

To understand the potential offered by the scientific perspective within CS, one must first appreciate the specific goals and methodologies that scientists (but not yet computer scientists) adopt as absolutely fundamental.

3 Seeking A Different Understanding

CS has been largely dominated by the engineering perspective. As a result of brilliant engineering, our discipline has generated a good number of industries, each revolving around stable, long-standing, complex, prevalently used computational tools. The engineering perspective can be characterized as seeking improvement, such as faster, more functional, more reliable.

CS has also successfully utilized the mathematical perspective. Complexity theory, asymptotic analysis, serializability theory, and other deep results have informed the study of computation in general and of the creation of software and hardware artifacts. The mathematical perspective can be characterized as seeking elegant formal structures and provable theorems within those structures.

The scientific perspective seeks a different kind of understanding: to explain phenomena encountered in nature. These explanations take the form of falsifiable *scientific theories* [Popper 1969] and *scientific laws* [Achinstein 1971]. The challenge to a scientist of any ilk is to reduce complex phenomena to simpler, measurable phenomena in such a way that the scientific community agrees that the essences of the original phenomena are captured. For example, Newton articulated the concept of "force" to explain the motion of the planets, John Nash provided an interpretation of human behavior as game playing, and E. O. Wilson and others dramatically changed ethology and ecology by introducing the ideas of energy budget and optimal foraging strategies.

The term "theory" in CS is a synonym for "formalism" or "concrete mathematics." When someone in CS does "theory," they are using the mathematical perspective. In logic, formal systems, and theorem proving, a "theory" is a purely formal object: a maximal set of assertions (strings) that can be deduced from axioms via inference rules. However, the term "theory" means something quite different in science.

Science seeks general, predictive theories that describe and explain observed phenomena. Such theories have four primary attributes.

- *Parsimony:* Each explains a variety of phenomena with a short, coherent explanation (Occam's razor), thereby reducing the number of independent phenomena.
- *Generality:* Each theory explains a wide range of phenomena.
- *Prediction:* Each anticipates future events, often to a high degree of accuracy.
- *Explanatory power:* Each provides a compelling underlying mechanism.

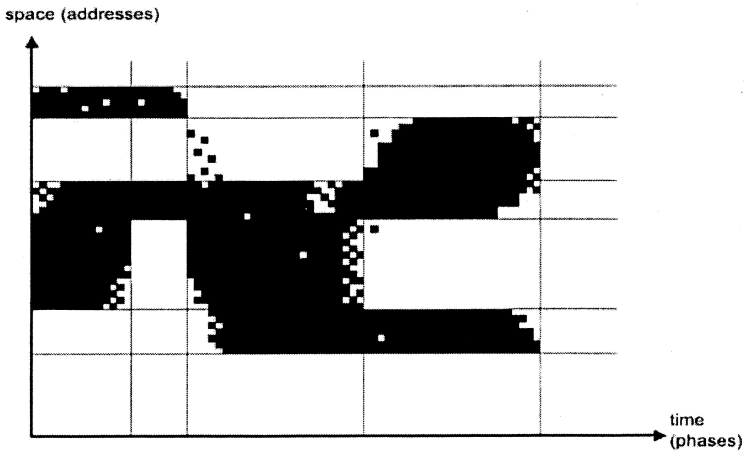

Figure 1: Locality-sequence behavior during program execution ([Denning 2005b, page 23])

Successful theories are also highly useful in engineering, because they can be employed to design mechanisms that ensure positive effects or avoid negative effects.

To illustrate these benefits, consider one of the few extant scientific theories of computation, Peter Denning's Theory of Locality [Denning 2005b]. It is useful to examine this theory in order to understand specifically how a scientific theory is radically different from the theorems of the mathematical perspective and the architectures and design rules of the engineering perspective. The locality theory also illustrates why it is critical to take the creator and user into account.

The locality theory arose from a study of the cost of managing page transfers between main memory and a much slower disk drive. "Because a bad replacement algorithm could cost a million dollars of lost machine time over the life of a system," this was a critical problem to solve. Denning noticed when measuring the relevant phenomenon, the actual page usage of programs, that there were "many long phases with relatively small locality sets" (see Figure 1); "each program had its own distinctive pattern, like a voiceprint." He found that nothing about the structure of memory or the processor executing instructions required or forced such locality sets. Instead, it was a clear, repeated pattern across many systems, users, and programs. Denning abstracted this observed behavior as follows. Define $D(x,t)$ as the *distance* from a processor to an object x at time t. If $D(x,t) \leq T$ for a specified distance threshold T, then x is defined to be in the *locality set* at time t. Locality theory states that if x is in the locality set at time t, then it is likely to be in the locality set at time $t + \delta$, for small δ, and so should not be removed from the cache.

The locality principle is inherently predictive and has been tested many, many times, for many distance functions. It is general: This theory applies to computational systems of all kinds: "in virtual memory to organize caches for address translation and to design the replacement algorithms, ... in buffers between computers and networks, ... in web browsers to hold recent web pages, ... in spread spectrum video streaming that bypasses network congestion" [Ibid, pages 23–24].

Where the locality principle becomes a truly scientific theory is in its explanatory power. Specifically, this theory identifies the source of the observed locality. One might think that perhaps this observed phenomenon of a locality set comes about because of the von Neumann architecture. But locality was originally observed not at the level of the instruction set, but between the main memory and the disk. And as just mentioned, locality has been observed in contexts unrelated to the von Neumann architecture. Instead, locality appears to originate in the peculiar way in which humans organize information:[1] "The mind focuses on

[1]This explanation by Denning of how locality works is reminiscent in some ways of Turing's explanation of his model of

a small part of the sensory field and can work most quickly on the objects of its attention. People gather the most useful objects close around them to minimize the time and work of using them." Psychology has described this process in quite precise terms and with compelling theories of its own: the persistent steering of attention, the chunking and limited size of short-term memory, the serial processing of certain kinds of cognition. This realization, this scientific theory of locality, thus holds in many disparate contexts, and has enabled specific engineering advances, including register caching, virtual memory algorithms based on efficiently computing the working set, and buffer strategies within DBMSes and network routers, thereby increasing the performance, scalability, and reliability of many classes of computational tools.

Another theory of computational tools is Vessey's Theory of Cognitive Fit, which states that "performance on a task will be enhanced when there is a cognitive fit (match) between the information emphasized in the representation type and that required by the task type" [Vessey 1991]. Both Locality Theory and Theory of Cognitive Fit are true scientific theories because they are parsimonious, general, predictive, and explanatory.

An important component of any scientific theory is a delineation of its *domain of applicability*, which defines the conditions under which nature (or computational tools) behave in a law-like way. Newtonian mechanics applies throughout the universe, but not at speeds approaching that of light nor at very small dimensions. As Thomas Kuhn [1996] emphasized, most scientists work within a paradigm with established theories. Much of what a scientist typically does relates to ascertaining the precise domain of a theory. But within that explicitly-stated domain, the theory should hold, and tests of that theory over phenomena with that domain should not fail. For scientific theories of computational tools, the domain might be a class of tool, e.g., a database management system (DBMS). Or the domain might be restricted, say to an attribute of a tool, e.g., to a DBMS utilizing cost-based query optimization. Or the theory might be more general, with a domain covering multiple classes of tools, as with the theory of cognitive fit. As another example, the established fact that caching works so well (whether in main memory, disks, networks, internet) is a testament that computational tools quite reliably exhibit locality. That said, the domain of applicability for locality does not include programs constructed explicitly to avoid locality.

4 Why Might Such Theories Hold?

Science desires theories that *endure*, i.e., that are not tied to a set of specific circumstances or to a particular time. Suppose that in studying phenomena related to a computational tool, one develops a theory that is reasonably general and parsimonious, that predicts accurately the observed phenomena, and has a compelling explanation for those phenomena. Why might such a theory be true? In particular, why might predictions when the theory is being tested in new ways in the future still be borne out?

The endurance of a theory of computational tools might arise in at least three somewhat different ways. First, it might be that the predictive power derives from some deep mathematical theorem that has yet to be proven or even stated. For example, perhaps Zipf's law [1932] (that for many tanked data, the relative frequency of the nth-ranked item is given by the Zeta distribution) will someday be shown to be a consequence of a yet-to-be stated mathematical theorem with an associated proof. The undiscovered theorem is what has caused predictions of this theory to hold. Specifically, the failure of a test of an hypothesis of such a theory might require an internal mathematical contradiction. There are possibly phenomena of computational tools

"machine," in which he appealed to how computing was done by (human) computers, an explanation which implicitly includes a notion of locality. "Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. ... Besides ... changes of symbols, the ... operations must include changes of distribution of observed squares. The new observed squares must be immediately recognisable by the computer [a human computer—ed.]. I think it is reasonable to suppose that they can only be squares whose distance ... does not exceed a certain fixed amount." [Turing 1937, pages 249–50]

that can be predicted and generalized into theories that are true because of yet-to-be-conceived mathematical truths.

A second, potent source of endurance and generality of scientific theories of computational tools is the very specific psychological abilities and limitations of humans. As emphasized above, locality in programs arises from "the peculiar way in which humans organize information.... The locality principle flows from human cognitive and coordination behavior...These behaviors are transferred into the computational systems we design" [Denning 2005b, page 24]. In Codd's Relational Theory, the relational data model (data as tables) and its associated tuple relational calculus derived in part from the syntactic structure of natural language queries. Indeed, the notion of relation, under the name of "predicate" goes back to the Greeks' attempts to identify the structure of logical arguments in order to lay down the rules (of inference) for correct reasoning. They realized, as logic does today, that there is something fundamental in the way that humans organize thought using predicates (relations) and their manipulation. The critical fixed point of all computational tools that we know about is that they were created and are used by *homo sapiens*.

A third source of endurance and generality of theories of computational tools is that of nature itself, independent of the role of humans in the construction and use of computational tools. Denning has argued that "computation and information processes have been discovered in the deep structures of many fields" [Denning 2007, page 13]; he gives as examples computation via DNA expression and computation via combining quantum particle interactions through quantum electrodynamics (QED). Just as there are laws of physics and chemistry and biology that govern nature, perhaps there are laws that govern computation in nature. Then computational tools would be subject to those laws and theories. Perhaps Newell and Simon's Physical Symbol System Hypothesis ("A physical symbol system has the necessary and sufficient means for general intelligent action") [Newell & Simon 1976] derives from such natural processes. Perhaps the observed preference by humans for locality also derives from natural processes of computation, in that there was no other way for us to think. (*That* would be an amazing discovery!) Or perhaps there are computational models in nature that are radically different from how humans process information.

5 Ergalics

The phrase "natural science of computational tools as a perspective of computer science" is verbose and awkward. The term "science of computer science" is shorter but still awkward. Hence, we use the term "ergalics" for this new science. It derives from the Greek word ergaleion ($\epsilon\rho\gamma\alpha\lambda\epsilon\iota\omega\nu$), translated by both Woodhouse's *English Greek Dictionary* and Liddell and Scott's *A Greek-English Lexicon* as "tool" or "instrument." This etymology is related to the Greek word ergon ($\epsilon\rho\gamma\omega\nu$), meaning "work," "job," or "task," and is the basis of the English words "ergonomics," an applied science of equipment design, "ergometer," an apparatus for measuring the work performed by a group of muscles, and the "erg," a CGS unit of work. These other terms emphasize the physical body; ergalics emphasizes computation. Some advantages of this term are it is a new word (Google it) and thus not overloaded with existing meaning, it is short, it is consistent with other scientific terminology, and it doesn't have negative connotations.

The goal of ergalics is to express and test scientific theories of computational tools and of computation itself and thus to uncover general theories and laws that govern the behavior of these tools in various contexts. The challenge now before the CS discipline is to broaden its reliance upon the mathematical and engineering perspectives and to embrace the scientific perspective.

The class of computational tools is broad and includes formalisms, conceptual devices, algorithms, languages, software systems, hardware, and computational infrastructure. Expanding on one of these categories, software systems include operating systems (e.g., Linux, Windows), database management systems (e.g., MySQL, Oracle), compilers and software development environments (e.g., Eclipse, gcc, Visual

Studio), GUIs (e.g., Java Swing, Windows), internet servers (e.g., Apache, Internet Information Server), browsers (e.g., Internet Explorer, Mozilla Firefox), and network protocols (e.g., DNS, TCP/IP). Note that in each case there exist both open source systems and proprietary systems, that each system involves a highly functional and quite complex interface, and that each system has been around for at least a decade. Also note that each kind of software system is associated with an annual one billion US$ industry and that each each of the categories can be enumerated and expanded, for CS has produced many significant, long-standing, mature computational tools.

Figure 2 illustrates the range of phenomena to be investigated within ergalics and for which scientific theories can be advanced and tested. A computational tool is first created (arc A) by a person or team of people given an identified task. This tool is then used by a person (arc B) to accomplish that task (arc C). Phenomena can be associated with each arc. Most CS research considers how to create and compose tools that accomplish a specific task (arc C). Such research considers such aspects as functionality (what does the tool actually do?), performance (what resources, e.g., CPU execution time, disk and network bandwidth, cache, main memory, disk space, does the tool require?), and reliability (to what extent does the tool accommodate failures of the hardware and software components it utilizes?). For example, the problem Denning first considered in his doctoral dissertation that ultimately lead to his Theory of Locality was the performance implications of disk reads, specifically the phenomenon of thrashing.

The study of an individual tool, while potentially quite insightful, is however only a small portion of the phenomena that can be investigated and the insights that are possible. One could study how that tool was influenced by human traits. (We argued earlier that locality theory ultimately concerned exactly that.) Viewed another way, a computational tool is a means to bridge the *computation gap* between the abilities of the user and the requirements of the task. Hence, one could study how tools are actually used and their effectiveness in order to discover enduring predictions and theories about such use. (It is important to recognize that some systems may be sufficiently *ad hoc* that there may not be interesting scientific theories that can be applied to them. However, for established, long-lived computational tools such as those listed above, it is likely that there is much to be gleaned from careful scientific analysis.)

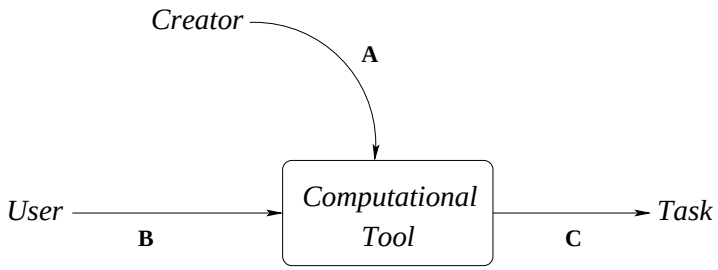

Figure 2: Computational tools

How does ergalics relate to Simon's "sciences of the artificial" [Simon 1996]? In terms of *methodology* and *goals*, the two are identical: both use the methodology of empirical science and have as the ultimate goal the articulation and testing of scientific theories. In terms of *subject of study*, Simon's sciences target "artifacts," which are things that humans construct to accomplish a task. The subject of study of ergalics is computation in nature *and* computational tools; the latter are artifacts but the former is not. We are in complete agreement with Simon and with Peter Denning [1995, 2005a, 2007], Peter Freeman and David Hart [2004], Walter Tichy [1998], and others who argue that legitimate science can extend beyond what people traditionally think of as "natural" science, to include the scientific study of artifacts as well. It is in terms of *epistemology* that the two differ the most. For Simon, the goal or task of an artifact is central; of primary interest is how the artifact accomplishes (or not) the goal. Ergalics concerns both the interaction of

the artifact and its goal as well as the artifact itself. For example, the locality exhibited by programs occurs independently of the design goal of the program; rather, locality arises through the innate information processing capabilities of its human creator. Ergalics similarly broadens its epistemology to embrace scientific explanations of the structure and behavior of computation occurring in nature, whether goal-directed or not.

Viewed in this way, Simon's distinction between "natural science" and "sciences of the artificial" does not apply to ergalics. The enduring ergalic theories are founded on as-yet undiscovered mathematical structure, on information-processing constraints on human thought and action, and on the occurrence of computation in nature itself. Ergalics is thus truly a natural science, not separate from the other sciences.

6 Asking Different Questions

A focus on computational tools and on computation itself elicits a wide range of relevant research questions. Figure 3 expands on the relevant phenomena. In this figure, the arcs identify interactions and influences from one component to another, each suggesting some overarching research questions within ergalics. One can ask, which tasks are desired (arc D) and how does the task influence both the tool and the use of the tool? (One could even ask, can a tool developed for one task be applied for a perhaps quite different task?) Such questions get at the core of a "science of design" [Freeman 2004]. A defining characteristic of computational tools is their capability of creating other computational tools (arc E). For example, lex and yacc create parsers, graphical query tools create multipage SQL queries, and sophisticated hardware and software development environments produce code fragments or generate complete applications or hardware given higher-level specifications. How are the computational tools we create limited or impacted by those we have available today? Do tools created by other tools exhibit the same characteristics as tools created directly by humans? For example, compiler-generated assembly code looks very different from human-generated assembly code: the former tends to be more bloated. This is also true of machine-generated high-level code (many programming language implementations now compile down to the C programming language and then invoke gcc), to the point where machine-generated C programs are considered a good torture test for C compilers. Interestingly, such machine-generated code exhibits locality, in part because modern architectures make non-locality so punitively expensive that any reasonable programmer would be careful to pay attention to locality considerations.

CS has also very effectively exploited the generality of computational tools by leveraging abstraction: higher-level tools exploit the functionality of lower-level tools. The *LAMP stack* (Linux-Apache-MySQL-Perl) consists of over 10 million lines of code [Neville-Neil 2008]. This approaches the intellectual complexity of the Saturn V rocket that took man to the moon (with three million parts), but it is to date much less well understood. The LAMP stack is a testament to the brilliance and sheer brute force required in the absence of a scientific foundation. What are the underlying structuring principles, general and predictive theories, and inherent limitations of such complex assemblages of individual tools, each itself a complex assemblage of sophisticated modules?

The availability of computational tools can change the learning and work and play of its users (arc F) and creators (arc G); witness the facile, ubiquitous use of social networking and instant messaging by our youth. Finally, tasks evolve, as new tools become available and as societal needs change, completing the social and cultural context within which the creator and user operate (arcs H and I) and thus indirectly impacting the evolution of tools and tool usage. Each of these arcs draws out an expansive range of phenomena to be studied, patterns to be identified, and the scientific theories to be uncovered as our understanding deepens.

As Newell, Perlis, and Simon emphasized, the science of computer science (ergalics) studies computational tools. Cohen agrees, "Unlike other scientists, who study chemical reactions, processes in cells, bridges under stress, animals in mazes, and so on, we study computer programs that perform tasks in en-

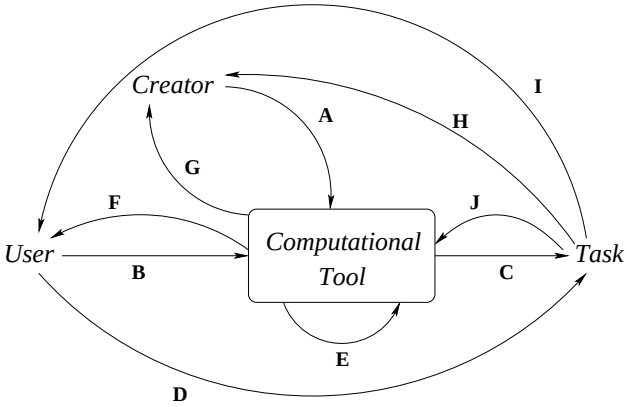

Figure 3: Computational tools, elaborated

vironments." [1995, page 2]. Whether studying a rat or a program, one must examine the behavior of the organism in context, as illustrated in Figure 4.

> "Whether your subject is a rat or a computer program, the task of science is the same, to provide theories to answer three *basic research questions*:
>
> - How will a change in the agent's structure affect its behavior given a task and an environment?
> - How will a change in an agent's task affect its behavior in a particular environment?
> - How will a change in an agent's environment affect its behavior on a particular task?" [Ibid, pages 3–4]

Note that the "organism" in Figure 4 is either the computational tool, with the user out of the picture or part of the environment, or the organism could be considered the user, with the computational tool being part of the environment, depending on which interaction of Figure 3 is under study.

A fourth basic research question is also relevant: How well does the agent perform the original task within the original environment? This can be viewed as another take on Figure 3.

7 Using Different Evaluative Strategies

The evaluative strategies used in other sciences suggest how we can evaluate our theories. Ergalics uses *empirical generalization*, in which understanding proceeds along two conceptual dimensions, as shown in Figure 5. As Paul Cohen [1995] has articulated, science (in general, and ergalics specifically) progresses (in the figure, on the y-axis) from studies of a particular system to statements about computational tools in general. So initial work might concern an aspect of a single program (e.g, a particular database query optimization algorithm in the context of a particular database management system), then generalize through studies of a class of systems (e.g., a study *across* several disparate DBMSs), to statements about computational tools in general (e.g., a theory that holds for rule-based optimizers, whether in DBMSs, AI systems, or compilers). This progression increases the domain of applicability, and thus the generality, of a theory.

Science (and ergalics) also progresses (in Figure 5, on the x-axis) from description of the phenomenon, to prediction, and eventually through causal explanation, within an articulated, thoroughly-tested theory.

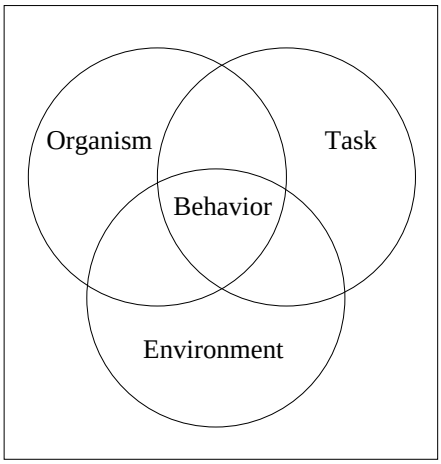

Figure 4: How the structure, task, and environment of an organism influence its behavior ([Cohen 1995, page 3])

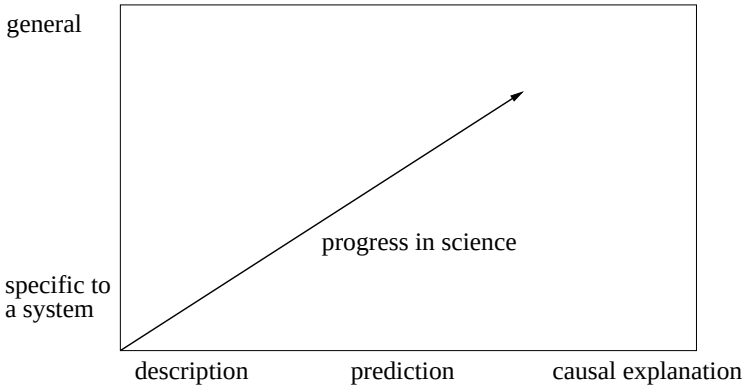

Figure 5: Empirical generalization ([Cohen 1995, page 5])

Consider the software tools listed in Section 2. Such prediction and causal explanations involve statements about accessibility, applicability, composability, correctness, extendability, functionality, performance, reliability, robustness, scalability. usability, and utility. Deep understanding involves being able to accurately predict such aspects of a computational tool and to articulate compelling causal explanations as to why those predictions hold up.

While *descriptions* of how these tools work are available, there is little about extant computational tools that can be scientifically *predicted* and very little *causal explanation.* CS in its first fifty years has restricted itself largely to the bottom-left quadrant of this space. We propose to expand the perspective radically in both dimensions.

Science proceeds from a combination of induction from observed data and deduction from a stated theory. The initial *theory construction phase* starts from observation of phenomena to tentative hypotheses. The goal here is to develop a *model* of the world (that is, within the explicitly stated domain of applicability) that can yield predictions that are are tested against data drawn from measurements of that world (see Figure 6).

The familiar process of debugging code can be viewed as an example of model testing. The *real world*

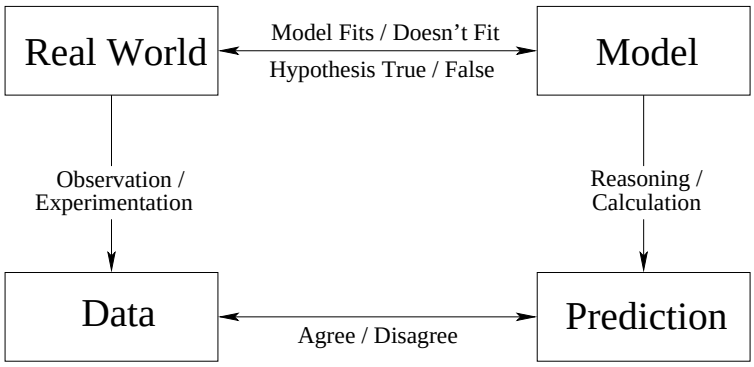

Figure 6: Model Testing ([Giere 1979, page 30])

consists of my program and its execution context: the compiler and the operating system and hardware on which the program runs. I test this program with some specified input data, as well as the specified correct output, which is what I hope the program will produce on this input data. I discover a flaw when the actual execution (the *observation/experimentation*) does not exactly correspond to the correct output. Through code inspection, I create my *model*: the line of code that I feel contains the flaw that produced the fault. I then form a *prediction* through *reasoning or calculation*: if I change that line of code, the regression test will now result in the expected output. My test of this hypothesis will tell me whether my model is an accurate reflection of (*fits or doesn't fit*) the world: does my prediction *agree or disagree* with the *data* actually produced when I reran the regression test? I could call this model a scientific theory, though it is not general at all (it applies only to this one program), it is not very predictive (it applies only to this one input data set), and it has weak explanatory power (it involves only the slice of the program involved in this regression test), thus placing this model towards the far bottom-left of Figure 5.

A scientific theory, its associated model, and predictions emanating from that model are tested until a more general theory emerges through insightful, creative induction. That theory is tested by deducing hypotheses, testing those hypotheses, and either confirming or rejecting them. Inevitably the theory is refined and its boundaries sharpened through further cycles of deductive testing and inductive refinement[2], as illustrated in Figure 7. This evolution of scientific theories occurs as a competition between competing models, with a range of evaluative tools to judge and select among them. The works of Ronald Giere [1979] and Paul Cohen [1995] provide useful tools (methodological and analytical) to move up the empirical generalization arrow of Figure 5 and to compare the relative strength of competing theories.

From prediction comes control. Improvement in our computational tools derives from the predictive power of our scientific theories. Ultimately, the value to society of a specific ergalic theory derives from the extent to which that theory provides engineering opportunities for improvement of a class of computational tools and explains the inherent limitations of that class of tools. For example, Denning's theory initially enabled efficient virtual memory systems, at a time when such systems were ad hoc and at times inadequate. His insight allowed us to see how programmers can adjust their approach to take advantage of locality, by, for example, using new kinds of hardware and software architectures. His theory also explained why the lack of sufficient main memory to hold a process's working set would inevitably result in poor performance.

[2]It should be acknowledged that this is a simplification; philosophers of science have argued for decades that this linear process is much more complex and that theories and questions reflect the biases and cognitive processes of the researchers, just as with their computational tools.

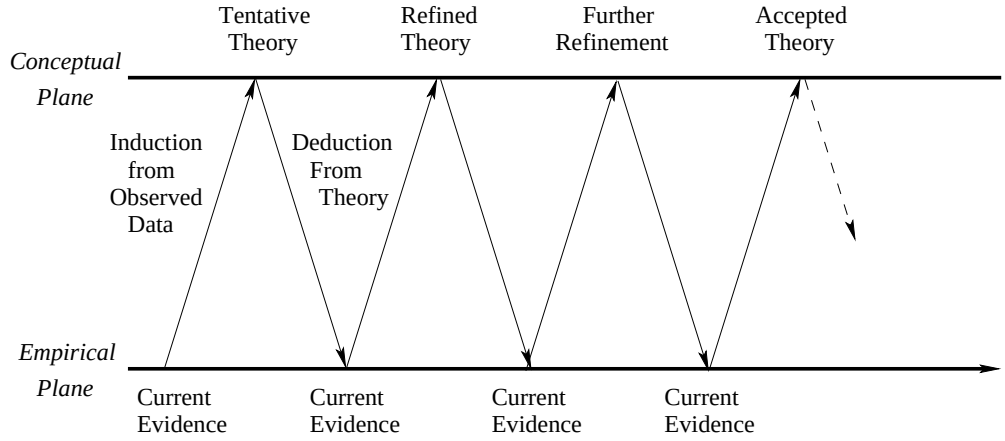

Figure 7: Theory construction, testing, and refinement ([Kohli 2008])

8 Interacting with Other Disciplines in Different Ways

To this point CS primarily has offered computational tools and computational thinking to the other sciences, so that scientists in those domains can advance their own knowledge.

With ergalics, the flow can also go the other way: computer scientists can utilize the insights, methodologies, experimental strategies, formalisms, and extant theories of other sciences to aid our understanding of computational tools. Specifically, (i) psychology can contribute directly, through its understanding of the structures and limitations of cognitive and non-cognitive thinking and of the challenging but often insightful methodology of user studies; (ii) other behavioral and social sciences can also provide insights into the social implications of human use of computational tools, such as how the internet is used; (iii) neurology, through such tools as functional MRI, can start to identify physical brain structures that influence the creation and use of tools; (iv) economics and sociology can provide both micro and macro behavioral clues, such as how trust in security and privacy mechanisms may be attained, and (v) physics, chemistry, and biology can provide evidence of computational processes in nature and possible theories and laws about those processes, which would also govern the behavior of computational tools.

The models and theories these other fields have developed are all getting at the basic research questions that Cohen articulated, concerning animals, organizations, systems, and (in ergalics) computational tools. It is likely that the models will be more similar than different, because most arise from similar methodological tools and support the same kinds of inference: prediction, generalization, control, and, occasionally, causal explanation. Ergalics can do what these other sciences have done and are doing: utilize the empirical generalization of Figure 5 to ask the questions implied by Figure 4 and to follow the general approach of theory construction, testing [Tichy 1998, Zelkowitz & Wallace 1997], and refinement of Figure 7.

The fact that ergalics utilizes a common scientific terminology with shared semantics and a common research methodology can have another real benefit. Peter Lee, currently chair of Carnegie Mellon University's Department of Computer Science, asks, why is it that "there isn't much computing research in the major core-science publications" [Lee 2008]. One possible reason is that CS does not as yet utilize the methodology and terminology of science, nor does it ask the questions of science, nor does it seek scientific theories. When CS participates in the market of enduring, technology-independent scientific theories (in this case, of computational tools), computing research may become more relevant for core-science publications.

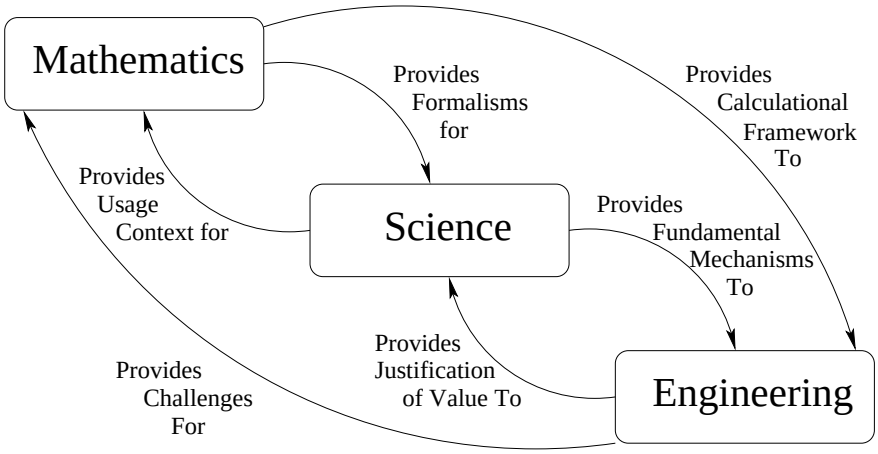

Figure 8: The interrelationship of the three perspectives

9 A New Direction

Ergalics seeks insights that are not based on details of the underlying technology, but rather continue to hold as technology inevitably and rapidly changes. It also seeks understanding of limitations on the construction and use of computational tools imposed by the nature of computation and by the specifics of human cognition. Ergalics provides an opportunity to apply new approaches, new methodological and analytical tools, and new forms of reasoning to the fundamental problems confronting CS and society in general. And it seeks to do so by bringing science into CS.

Science provides a specific methodology which has been extremely productive and beneficial in other sciences. The development of scientific theories in CS can produce new insights and better outcomes than restricting ourselves to the mathematical and engineering perspectives. Denning's recent assessment is hopeful, but as yet unrealized: "The science paradigm has not been part of the mainstream perception of computer science. But soon it will be" [Denning 2005a, page 31].

As Jeannette Wing stated in a talk [Wing 2008] at Stanford on May 21, 2008, the fundamental question of the Network Science and Engineering (NetSE) initiative is, "Is there a *science* for understanding the complexity of our networks such that we can *engineer* them to have predictable behavior?" (emphases in the original). Ergalics generalizes this challenge to *computational tools* and to *computation* itself, manifesting scientific theories across CS disciplines and enabling the engineering advances enabled by such theories. To achieve this result, we need to restructure the educational and scholarly ethos of CS to also encourage and enable the scientific perspective.

Bill Wulf has observed [1995] that "Young as we [the discipline of CS] are, I think we really don't have a choice to be science or engineering; we are science and engineering, and something more, too." Figure 8 shows how the three perspectives, with science in the middle, symbiotically relate. Mathematics is often useful in providing formalisms to succinctly express theories (thus, aiding parsimony) and fit them in with other models of nature (generality). It also gives us a way to think about deep structure, and thus helps us to expose that structure. Science can be useful to engineering by explicating the underlying causal mechanisms (prediction and control). Similarly, engineering reveals behavior (phenomena) that science can use to construct new theories, and science provides needs for new formalisms and theorems.

As an example from aeronautics, the mathematics perspective has developed non-linear differential equations, in particular the Navier-Stokes equations, which can be used in the momentum equations for fluids such as air. The scientific perspective has used such formalisms to express thermodynamic laws, such

as the perfect gas equation of state, relating pressure, density, and temperature. This physical theory allows one to accurately predict the viscous flow of a fluid around an object and the impact of changing temperature and pressure. The engineering perspective uses, for example, the Reynolds number, a notion grounded in these scientific laws, to scale data from wind tunnel experiments on wing shapes to their real-life counterparts.

This structure can be applied to CS, with concrete mathematics [Graham, Knuth, and Patashnik 1994] and complexity theory to the left, ergalics in the center, and most of conventional CS on the right. (In fact, an important objective of this essay has been to characterize exactly what that center component consists of, what a science of computer science would entail, in terms of its objects of study, its methodology, and its emerging understanding.) An increased emphasis on ergalics will bring balance to this interrelationship, providing a solid foundation upon which to make great strides in the engineering of computational tools, thereby dramatically increasing their performance, efficacy, and reliability.

10 Benefits

How might the articulation of scientific theories of computational tools benefit society? This question can be approached in several ways.

First, looking back over the last three hundred years, it is clear that many if not most of the technologies that emerged were *preceded* by a deep understanding afforded by scientific theories and mathematical formalisms and theorems. Consider the $10 GPS chip in all cell phones. This advance required Newton's Laws of Motion and the mathematics of celestial navigation to place the GPS satellites in geosynchronous orbit, Bohr's Atomic Theory to enable the Cesium atomic clocks on those satellites, Electromagnetic Theory and Maxwell's equations to enable the efficient transmission and reception of weak radio signals, and Quantum Mechanics and Shrödinger's equations to enable the construction of tiny switching transistors within the GPS chip. A similar analysis for other engineering breakthroughs over the last few centuries emphasizes that the discovery and elaboration of a scientific theory and its associated mathematical underpinning and deep understanding often *enabled* the engineering advance.

Second, an historical analysis of CS over its fifty-year lifetime also shows the critical role of scientific theories. In the mid-1960's, operating systems were at a crossroads. Companies "were reporting their systems were susceptible to a new, unexplained, catastrophic problem they called thrashing. Thrashing seemed to have nothing to do with the choice of replacement policy. It manifested as a sudden collapse of throughput as the multi-programming level rose" [Denning 2005b, page 21]. The ergalic theory of locality and its understanding of the central role that the working set played was followed by appropriate page replacement and job scheduling algorithms that solved this catastrophic problem.

However, the true benefits of a science of computation, of to-be-articulated ergalic theories, are impossible to predict, because scientific theories, by virtue of the deep understanding they encapsulate, are transformational and disruptive. Given the well-established connection, in some cases causal, from scientific theory to engineering innovation, the discovery of ergalic theories may be a necessary prerequisite for some innovations. And it is probable that the solutions of many of the technological problems related to our use of computational tools will require engineering innovations that are predicated on ergalic theories still to be articulated. In consideration of the advances of information technology that have resulted from the interplay of two primary perspectives, mathematics and engineering, might the incorporation of a *third* perspective accelerate innovation, energize and intrigue students, and increase public support?

11 An Opportunity

People use computational tools. People also construct these tools. Computational tool construction and use is one of the ways that humans are unique. The tools that humans produce and the ways that they use such tools are profoundly affected by the way that humans think. Ultimately, understanding computational tools enables us to build better tools, and helps us to understand what makes us human.

Computation also appears to be a fundamental process in nature. If so, the scientific perspective affords a way to better understand our world.

Unlike established sciences, where many if not most of the fundamental theories have already been discovered [Horgan 1996, Horgan 2004] (though much elaboration remains), the theories of ergalics are still out there, just waiting to be uncovered. Who will discover the CS equivalents of Einstein's theory of relativity, of Mendel's theory of heredity, of Darwin's theory of evolution, of Festinger's theory of cognitive dissonance, of Pauling's theory of chemical reactions?

12 Acknowledgments

I offer my sincere thanks to those who have commented on previous versions of this essay, bringing relevant connections and viewpoints: Greg Andrews, Kobus Barnard, Carole Beal, Merrie Brucks, Faiz Currim, Sabah Currim, Paul Cohen, Saumya Debray, Peter Denning, Peter Downey, Ian Fasel, John Hartman, Laura Haas, John Kececioglu, Leonard Kleinrock, Lester McCann, Clayton Morrison, Frank Olken, Ben Shneiderman, Maria Zemankova, and Beichuan Zhang. I especially appreciate the hours of discussion on these topics that many of these people each spent with me.

References

[Achinstein 1971] P. Achinstein, **Law and Explanation**, Clarendon Press, Oxford, 1971.

[Aicken 1984] F. Aicken, **The Nature of Science**, Heinemann Educational Books, London, 1984.

[Chalmers 1999] A. F. Chalmers, **What is this thing called Science?** Third Edition, Hackett Publishing Company, 1999.

[Cohen 1995] P. Cohen, **Empirical Methods for Artificial Intelligence**, MIT Press, 1995.

[Davies 1973] J. T. Davies, **The Scientific Approach**, Academic Press, New York, 1973.

[Denning 1995] P. J. Denning, "Can There Be a Science of Information?", *ACM Computing Surveys* 27(1):23–25, March 1995.

[Denning 2005a] P. J. Denning, "Is Computer Science Science?", *CACM* 48(4):27–31, April 2005.

[Denning 2005b] P. J. Denning, "The Locality Principle," *CACM* 48(7):19–24, July 2005.

[Denning 2007] P. J. Denning, "Computing is a Natural Science," *CACM* 50(7):13–18, July 2007.

[Freeman 2004] P. Freeman and D. Hart, "A Science of Design for Software-Intensive Systems," *CACM* 47(8):19–21, August 2004.

[Giere 1979] R. N. Giere, **Understanding Scientific Reasoning**, fourth edition, Harcourt Brace, Fort Worth, TX, 1979.

[Graham, Knuth, and Patashnik 1994] R. L. Graham, D. E. Knuth, and O. Patashnik, **Concrete Mathematics: A Foundation for Computer Science**, second edition, Addison-Wesley, Reading, MA, 1994.

[Horgan 1996] J. Horgan, **The End of Science**, Helix Books, Addison-Wesley, Reading, MA, 1996.

[Horgan 2004] J. Horgan, "The End of Science Revisited," *IEEE Computer* 37(1):37–43, January 2004.

[Kohli 2008] A. K. Kohli, "Theory Construction in Marketing," talk at the University of Arizona, April 2008.

[Kuhn 1996] T. Kuhn, **The Structure of Scientific Revolutions**, University of Chicago Press, 1996.

[Lee 2008] P. Lee, "Science and Nature: Where's the Computing Research?" Computing Community Blog, September 12, 2008, `http://www.cccblog.org/2008/09/12/science-and-nature-wheres-the-computing-research/`, viewed September 18, 2008.

[Neville-Neil 2008] G. V. Neville-Neil, "Code Spelunking Redux," *CACM* 51(10):36–41, October 2008.

[Newell, Perlis, and Simon 1967] A. Newell, A. J. Perlis, and H. A. Simon, "Computer Science," letter in *Science* 157(3795):1373–1374, September 1967.

[Newell & Simon 1976] A. Newell and H. Simon, "Computer Science as Empirical Inquiry: Symbols and Search," *CACM* 19(3):113–126, March 1976.

[Popper 1969] K. R. Popper, **Conjectures and Refutations**, Routledge and Kegan Paul, London, 1969.

[Simon 1996] H. A. Simon, **Sciences of the Artificial**, Third Edition, MIT Press, Boston, MA, 1996.

[Tichy 1998] W. F. Tichy, "Should Computer Scientists Experiment More?," *IEEE Computer* 31(5):32–40, May 1998.

[Turing 1937] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society* s2-42(1):230-265, 1937.

[Vessey 1991] I. Vessey, "Cognitive Fit: A Theory-based Analysis of Graphs vs. Tables Literature," *Decision Sciences* 22(2):219–240, March 1991.

[Wing 2008] J. Wing, "Network Science and Engineering: Call for a Research Agenda," talk at Stanford University, May 21, 2008. `http://www.cra.org/ccc/docs/stanford.pdf`, viewed September 12, 2008.

[Wulf 1995] W. A. Wulf, "Are We Scientists or Engineers?", *ACM Computing Surveys* 27(1):55–57, March 1995.

[Zelkowitz & Wallace 1997] M. V. Zelkowitz and D. R. Wallace, "Experimental Validation in software engineering," *Information and Software Technology* 30:735–743, 1997.

[Zipf 1932] G. K. Zipf, **Selected Studies of the Principle of Relative Frequency in Language**, Harvard University Press, 1932.